Towards Optimal Power Management: Estimation of Performance Degradation due to DVFS on Modern Processors

Hrishikesh Amur Karsten Schwan Milos Prvulovic

College of Computing, Georgia Tech
amur@gatech.edu {schwan, milos}@cc.gatech.edu

ABSTRACT

The alarming growth of the power consumption of data centers coupled with low average utilization of servers suggests the use of power management strategies. Such actions however require the understanding of the effects of the power management actions on the performance of data center applications running on managed platforms. The goal of our research is to accurately estimate power savings and consequent performance degradation from DVFS and thereby better guide the optimization of a performance/power metric of a platform. Towards that end, this paper presents precise performance and power models for DVFS strategies. Precise models are attained by better modeling the performance behavior of modern out-of-order processors, by taking into account, for instance, the effects of cache miss overlapping. Models are validated using benchmarks from the SPEC CPU2006 suite, which show that the observed degradation always falls within the predicted bounds. Also, the upper bound degradation estimates were up to 43% less than those due to a linear degradation model which allows for the aggressive use of DVFS.

1. INTRODUCTION

The growth in power consumption of modern data centers has been well documented. While different power management mechanisms are available to reduce power consumption in times of low utilization, their aggressive use is difficult because their effect on application performance is not well understood. Consider a typical data center where the average CPU utilization is around 20-30%[2]. Utilization is low in part because data center operators are conservative about provisioning for peak activity and in light of the business risks associated with violations of SLAs. To address these issues, we must better understand the effects of active power management on application performance, e.g., of power scaling of otherwise underutilized machines [8]. Stated more precisely, if we can bound the degradation of application performance, active power management can be done so as to continue to meet the SLAs of applications. Our research seeks to provide such bounds for modern processors and representative data center applications.

Under low CPU utilization, multiple processor power management actions can be taken on a platform running some workload. Unfortunately it may not be clear which action optimizes the platform's performance/power ratio. Consider, for simplicity, a single-threaded application running on a CPU of an SMP machine as shown in Figure 1a. If the

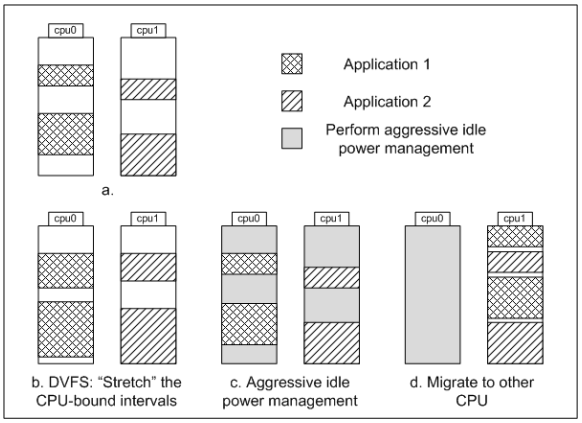


Figure 1: Methods for Power Management

CPU utilization is low, there is an opportunity for processor power management on that particular CPU. Of the multiple ways of saving power, the first and popular method, shown in Figure 1b, could be to scale down the frequency and voltage of the CPU proportional to the CPU utilization. This essentially 'stretches' the execution time of the processes and saves power by running at a lower performance state. A second method, shown in Figure 1c, could be to run the application at the highest power state during its CPU-bound part. This maximizes the idle time available for the CPU to switch to a power saving deep idle state. A third method, represented in Figure 1d, could be to migrate the application to another CPU/package which is almost completely utilized and free the current CPU/package completely so that it can be put in to a deep idle state. Addressing the question – which of these strategies to use – our approach consists of the following steps:

1. To fully understand the reasons for performance degradation in each scenario as well as the potential energy savings and quantify them.
2. To arrive at the optimal decision by optimizing a metric based on performance/power using data obtained from the previous steps.

There has been significant past work on exploiting power savings 0 while minimizing performance degradation. As observed by [9, 4], DVFS has little effect on performance for a highly memory-bound application that spends a significant amount of time waiting on DRAM accesses. When using an aggressive idle power management, apart from the predictable degradation due to transition time, transitioning

to a deep idle state might show unexpected degradation of performance due to loss of cache contents since deeper idle states save power by flushing and then turning off cache levels progressively. The performance degradation then, would be dependent on the cache-resident working set size of the application [1], so a strategy purely based on the transition times would not yield accurate results. Srinivasan, *et al* [10] point out that, when consolidating tasks on to a smaller number of CPUs, it is also important to migrate timer-handling and interrupt-handling responsibilities to fully reap the benefits of consolidation.

Using the insights reported above, our research is developing a model that provides quantitative estimates of the performance degradation associated with the methods for power savings reviewed above. Our first step in this research considers DVFS, providing upper bounds on degradation that can then be used to guarantee operation within the limits of application SLAs. Technical contributions include the ability to calculate bounds on degradation for modern out-of-order processors, in contrast to prior work by Choi, *et al* [4] who use on/off-chip execution time ratios to drive DVFS policies in the embedded space. In addition:

1. We accurately estimate the bounds on potential performance degradation from DVFS, using a performance model that incorporates the significant effects of out-of-order execution, a standard feature on modern processors. We find that ignoring these effects could cause prediction errors of up to 20%.

2. We evaluate our model using benchmarks from the SPEC CPU2006 suite and show that the observed degradation falls within the calculated bounds. We also show that our method decreases the upper bound predicted value by up to 43% as compared to linear degradation models.

3. We model potential power savings by actual power measurements, accounting for the effects of processor topology. This information along with the predicted performance degradation makes it possible to compute a performance/power metric that can then be used as a basis for comparing different power management strategies.

2. BACKGROUND AND DEFINITIONS

2.1 Performance Degradation

The performance degradation d_S of an application due to some power management strategy S is defined as:

$$d_S = \frac{t_S - t_0}{t_0} \tag{1}$$

Where, t_S is the total execution time with strategy S and t_0 is the total execution time when running at maximum performance. The remainder of this paper differentiates observed quantities from others, using the *bar* as in $\bar{U}$ to indicate observed (i.e., not calculated) for clarity.

2.2 T_{onchip} and $T_{offchip}$

The CPU utilization of an application consists of: (i) instructions that execute on the CPU or memory instructions that hit in any of the cache levels up to the last level on-chip cache(LLOC), termed T_{onchip}; (ii) the other fraction of CPU utilization that consists of memory accesses that hit in cache levels lower than the LLOC or progress to the DRAM, termed as $T_{offchip}$. This makes it clear that the T_{onchip} fraction is precisely the part of the CPU utilization that is affected directly by degradation due to DVFS. Thus for an application s,

$$T_{onchip}(s) = t_{CPU} + \sum_{i=1}^{N_{LLOC}} t_{L_i} \tag{2}$$

$$T_{offchip}(s) = \sum_{i=N_{LLOC+1}}^{N_{LLC}} t_{L_i} + t_{MEM} \tag{3}$$

$$\bar{U}(s) = T_{onchip}(s) + T_{offchip}(s) \tag{4}$$

Where $\bar{U}(s)$ the observed CPU utilization and t_{CPU} , t_{MEM} and t_{L_i} are the times per second spent executing on the CPU, waiting for memory accesses to complete and the hitting in cache level L_i respectively and N_{LLOC} and N_{LLC} are the levels of the last-level on-chip cache and last-level cache. Since these terms cannot directly be measured on most platforms we must estimate T_{onchip} and $T_{offchip}$. Consider, for example, a memory configuration with L_1 and L_2 on-chip caches. In this case, the value of T_{onchip} will depend on the distributions of the times spent executing on the CPU, hitting in L_1 and L_2 caches respectively. These values cannot be derived from directly observable performance events like cache misses because there exists considerable parallelism in the servicing of misses. For example, the time taken to service N L_1 misses can vary significantly depending on the degree of overlap of the misses. Therefore these effects clearly cannot be ignored.

3. PERFORMANCE DEGRADATION

The performance degradation of an application can be accurately predicted if the T_{onchip} and $T_{offchip}$ of the application are known. To empirically derive this relation we profile the machine with a synthetic benchmark. The benchmark models a throughput-based application for which the T_{onchip} and $T_{offchip}$ can be set. By making multiple runs with different values of T_{onchip} and $T_{offchip}$ we measure the performance degradation experienced when changing the frequency of the CPU from f_0 to $f, (f \leq f_0)$. From this empirical data we derive the expression for percentage degradation d in terms of T_{onchip} and $T_{offchip}$ where d:

$$d = \alpha_{f_0,f} T_{onchip} + \beta_{f_0,f} T_{offchip} \tag{5}$$

As expected, $\alpha_{f_0,f} \approx f_0/f - 1$ and $\beta_{f_0,f} \approx 0$ which means that the linear degradation affects only the on-chip execution. The T_{onchip} and $T_{offchip}$ values for an application can be quite difficult to determine since most processors do not allow the required quantities to be directly measured using performance counters. However almost all processors support the measurement of (i) the total number of instructions completed and (ii) the cache misses experienced at different levels of the cache. We use these facts as explained next.

3.1 Using LLOC misses

To estimate $T_{offchip}$ of an application s, consider that the sole cause of LLOC misses is the off-chip execution. Therefore, there is a strong correlation between the number of

Benchmark	T_{onchip}	$T_{offchip}$	Residency[1]	Overlap[2]
s_{LLOC+1}	0	1.0	$LLOC+1$	Maximal
s_{DRAM}	0	1.0	DRAM	No
s_{peak}	1.0	0.0	CPU/L_1	Maximal
s_{LLOC}	1.0	0.0	$LLOC$	No

Table 1: Synthetic benchmarks for boundary cases

LLOC misses and $T_{offchip}$:

$$\bar{x}_{LLOC} = x(s)T_{offchip}(s) \quad (6)$$

where, $\bar{x}_{LLOC}$ is the number of data LLOC misses and $x(s)$ denotes the number of LLOC misses per second that would have been seen for application s had $T_{offchip}(s) = 1$. To calculate $T_{offchip}(s)$ we obtain an estimate for $x(s)$ using bounds obtained from profiling the memory hierarchy with synthetic benchmarks. Estimating $x(s)$ however, presents some interesting problems. Consider the situation where, for a given number of LLOC misses, all the misses are serviced at level $N_{LLOC}+1$. Then the time taken(and the hence the $T_{offchip}$) would be much less than the case where all the misses go down to DRAM. We call this the effect of *residency*. It seems that it is at least possible, if not expensive, to determine the value of $x(s)$ exactly by measuring the cache misses at each level between N_{LLOC} and DRAM and then calculate a weighted average. But memory hierarchies associated with modern processors are sophisticated enough to allow significant overlapping of misses at each level of the hierarchy. The resulting problem is that the time spent in each level can no longer be derived from performance events like the cache misses at each level because the degree of overlapping of the misses could change the time significantly. This is called the effect of *overlap*. To address these issues, we find the two extreme cases and express both effects, residency and overlap, by using an *off-chip overlap factor* $\theta(s)$. It will be seen in this and later sections that residency and overlap are recurring themes that affect T_{onchip} and $T_{offchip}$ in a significant manner.

The synthetic benchmarks used, s_{LLOC+1} and s_{DRAM}, are shown in Table 1. The lower bound on $T_{offchip}$ is obtained from using the application s_{LLOC+1} because all memory accesses hit in level $N_{LLOC}+1$ and there is no dependence between the accesses, which allows for maximal overlap. The number of LLOC misses seen for application s_{LLOC+1} is termed x_{LLOC+1}. The upper bound is obtained from the application s_{DRAM}, where all accesses go to DRAM and there is explicit dependence between the accesses which prevents overlap. The number of LLOC misses seen for application s_{DRAM} is termed x_{DRAM}. Figure 2 shows the two synthetic benchmarks on a platform which has no off-chip caches, so $N_{LLOC} = N_{LLC}$. Therefore, since this case merely represents the presence vs. absence of overlap, the significant effects of overlap can be seen which clearly cannot be ignored.

The constants x_{LLOC+1} and x_{DRAM} obtained by profiling with the synthetic benchmarks are characteristics of only the memory hierarchy and therefore for any application s, $x(s)$ is a weighted average of x_{LLOC+1} and x_{DRAM}:

$$x(s) = \theta(s)x_{LLOC+1} + (1-\theta(s))x_{DRAM} \quad (7)$$

[1]Shows which level of the cache the memory accesses hit in.
[2]Maximal overlap corresponds to completely independent memory accesses and no overlap to fully dependent ones.

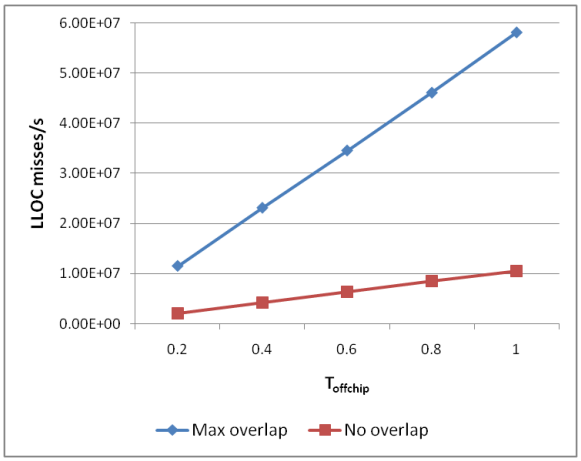


Figure 2: Variation of LLOC misses with $T_{offchip}$ to show the effect of overlap with s_{LLOC+1} and s_{DRAM}

where, $\theta(s) \in [0,1]$ is the *off-chip overlap factor*. From Equations 6 and 7, we derive:

$$\theta(s) = \frac{\bar{x}_{LLOC} - T_{offchip}(s)x_{LLOC+1}}{T_{offchip}(s)(x_{DRAM} - x_{LLOC+1})} \quad (8)$$

The *off-chip overlap factor* is a measure of both the overlap that exists among the off-chip cache levels and memory as well as the distribution of accesses hitting in each level.

3.2 Using the Total Instructions Completed per Second(IPS)

The intuition behind using the total IPS is that it consists of instructions that were completed on-chip and instructions(memory accesses) that went off-chip. This is useful because the instructions executed in the two cases are proportional to the T_{onchip} and $T_{offchip}$ respectively. By dividing the observed IPS into on-chip and off-chip instructions, we can hope to estimate T_{onchip} and $T_{offchip}$. We point out that the effects of residency and overlap, as seen before, also make the estimation of the on-chip and off-chip execution difficult.

To calculate the contribution to the total IPS from the off-chip execution, $i_{offchip}(s)$, we follow a similar procedure as before to obtain the bounds. So i_{LLOC+1} and i_{DRAM} represent the IPS for applications s_{LLOC+1} and s_{DRAM}. We argue that the *off-chip overlap factor* $\theta(s)$ calculated for LLOC misses can also be used as the factor in the calculation of the convex combination of i_{LLOC+1} and i_{DRAM}, since the instructions that contribute to the off-chip execution of the application are the very same instructions that were counted as LLOC misses. So had the application s run with $T_{offchip}(s) = 1$, the IPS would have been:

$$\hat{i}_{offchip}(s) = \theta(s)i_{LLOC+1} + (1-\theta(s))i_{DRAM} \quad (9)$$

Therefore the IPS due to off-chip execution for application s when running normally with $T_{offchip} = T_{offchip}(s)$ is:

$$i_{offchip}(s) = \hat{i}_{offchip}(s)T_{offchip}(s) \quad (10)$$

It is not possible to directly measure the contribution to the observed IPS from on-chip execution. This is because the exact distribution of the times spent executing on the CPU and hitting in on-chip caches $L_1, ..., L_{N_{LLOC}}$ is difficult to estimate. Further, even if the hit/miss counts for each

level of the cache hierarchy were known, the overlap between the misses makes it impossible to measure the *times* spent hitting in each level. Therefore, we use benchmarks shown in Table 1 to establish bounds on T_{onchip}. The lower bound on T_{onchip} will occur when the IPS of the system hits the peak value i_{peak}. The upper bound occurs in the case of application s_{LLOC} where the on-chip execution is entirely due to memory access instructions that hit in the LLOC with no overlap of memory accesses, and the IPS in this case is i_{LLOC}. Therefore for any application s the expected IPS had it run with $T_{onchip} = 1$ would be a weighted average of i_{peak} and i_{LLOC},

$$\hat{i}_{onchip}(s) = \lambda(s) i_{peak} + (1 - \lambda(s)) i_{LLOC} \tag{11}$$

where, $\lambda(s) \in [0, 1]$ is the *on-chip overlap factor*. The contribution to the observed IPS for application s from the on-chip execution running with $T_{onchip} = T_{onchip}(s)$ is:

$$i_{onchip}(s) = \hat{i}_{onchip}(s) T_{onchip}(s) \tag{12}$$

Also,

$$\bar{i}(s) = i_{onchip}(s) + i_{offchip}(s) \tag{13}$$

where $\bar{i}(s)$ is the observed IPS for application s. From Equations 8,10,11,12 and 13 we derive an expression for the $T_{offchip}$ of application s:

$$T_{offchip}(s) = \frac{\acute{x}\,(i_{onchip}(s) + i_{DRAM} - \bar{i}(s)) + \acute{i}\bar{x}(s)}{i_{onchip}(s)\acute{x} + x_{DRAM}\acute{i}} \tag{14}$$

where, $\acute{x} = x_{LLOC+1} - x_{DRAM}$ and $\acute{i} = i_{LLOC+1} - i_{DRAM}$. We estimate the bounds on $T_{offchip}$ by using the fact that $\lambda(s) \in [0, 1]$. To calculate the upper bound on $T_{offchip}$, we set $\lambda(s) = 1$ (or $i(s) = i_{peak}$) and setting $\lambda(s) = 0$ (or $i(s) = i_{LLOC}$) yields the lower bound for the $T_{offchip}$. We use Equation 4 to obtain corresponding T_{onchip} values and Equation 5 to evaluate the bounds for degradation.

4. POTENTIAL SAVINGS IN POWER

To determine the utility of a power management strategy, one must estimate the expected savings in power. One might expect the system power to decrease with a decrease in P-state, but the reality is that the topology of the processors plays a significant part. For example, multi-core platforms, such as the Xeon 5100 series[6], have hardware logic that coordinates between the voltage levels requested by each of the cores on the chip and selects the highest one for the processor package. To take into account the topology of system processors, we profile the system by measuring the system power for different configurations of the performance states for the CPUs. Table 2 provides the power profile associated with different configurations for our platform. We can see, for instance, that there is no advantage in reducing the P-state of CPU_1 if CPU_0 is at a higher P-state. Although the profiled machine is not very power efficient, the measurements demonstrate the importance of considering the topology of the system when building performance models. Power data is obtained using an Extech 380801 power analyzer which allows for out-of-band measurements.

Processor P-states				Power(W)
$Package_0$		$Package_1$		
CPU_0	CPU_1	CPU_2	CPU_3	
P0	P0	P0	P0	267
P0	P2	P0	P2	267
P0	P0	P1	P1	258
P0	P0	P2	P2	252
P1	P1	P1	P1	248
P1	P2	P1	P2	248
P1	P1	P2	P2	242
P2	P2	P2	P2	234

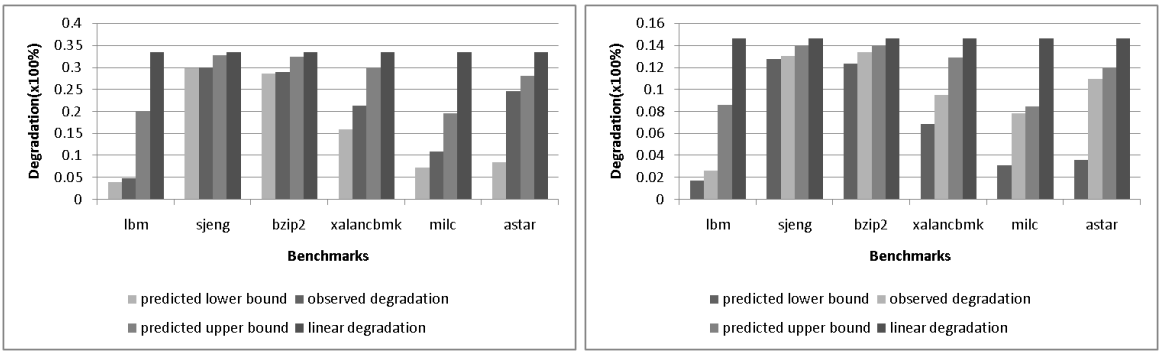
Table 2: System power consumption for some configurations of P-states.

5. RESULTS

Experimental evaluations are conducted on a dual-core double-socket Xeon(Woodcrest) processor. The processor provides three P-states at frequencies 2.67GHz, 2.33GHz and 2.00GHz. The system has 8GB of memory. The cache hierarchy consists of a 32KB L_1 instruction and data caches and a 4MB L_2 cache. We use benchmarks from the SPEC CPU2006 suite to evaluate our method. The benchmarks are selected on the basis of the presence of off-chip execution. In the absence of or with little off-chip execution the performance degradation from decreasing the frequency from f_0 to f becomes equal or very close to $f_0/f - 1$ which is easy to predict. For the selected benchmarks, we estimate the bounds on performance degradation for the benchmarks by decreasing the P-state of the CPU from P0 to P1 and P2 and compare the predictions against the actual performance degradation. The two cases are shown in Figure 3(a) and Figure 3(b) respectively. It can be seen from the graphs that the observed degradation for both cases falls within the predicted bounds for all the benchmarks. This is result is important for several reasons. Firstly, it validates the performance degradation prediction model and provides an upper bound for degradation which may be used for operation within application SLAs. Secondly, if we compare our model to linear degradation models where the degradation is assumed to be equal to $f_0/f - 1$, we see up to 43% reduction in the upper bound of the predicted degradation. Thirdly, the large difference in certain cases(up to 20%) between the predicted lower and upper bounds implies that failing to account for the overlap could cause estimation to be wrong by as much. To calculate the performance/power metric for DVFS strategies we use the predicted value of performance degradation with power values obtained from the power profile generated for the system. This can then be used for comparison with other strategies.

6. RELATED WORK

DVFS support is now commonly available on most modern processors. Various policies have been proposed in the past and these can be divided into static and dynamic policies depending on when decisions are made. Hsu and Kremer [5] study static compiler techniques while dynamic techniques using DVFS are explored by Wu *et al* [11], Nathuji *et al* [8]. Another approach to dynamically setting DVFS performance levels is to use Performance Monitoring Units(PMU) to detect when it is possible to achieve sublinear performance degradation. This technique has since been implemented on a number of platforms. Bellosa, *et al.* pioneered the development of energy models for the XScale architecture [3] and Isci, *et al.* build per-component power models



(a) $f_0 = 2.67$GHz and $f = 2.00$GHz (b) $f_0 = 2.67$GHz and $f = 2.33$GHz

Figure 3: Accuracy of prediction of degradation for CPU2006 benchmarks

based on observed activity [7] for the Pentium 4. Choi, *et al.* explored the idea of using on/off-chip execution ratios to effect DVFS strategies [4]. Rajamani, *et al.* use a heuristic based on the stalled time per instruction to decide the memory-boundedness of an application [9]. Xie, *et al.* propose algorithms to determine the upper bounds on energy savings [12]. To our knowledge, however, previous work using on/off-chip execution time ratios to make DVFS policy decisions either assumes that out-of-order execution has minimal impact or directly measures the effect for simpler cache structures. In contrast, our model *quantifies* the effects of the overlap using a completely general cache/memory hierarchy.

7. CONCLUSION AND FUTURE WORK

By determining the bounds on performance degradation associated with power management techniques, it becomes possible to reduce platform power consumption without compromising application SLAs. In this paper, we take steps towards this goal by constructing a performance degradation model for DVFS. It differs markedly from previous models for embedded systems because of the more complex nature of modern server architectures. For instance, the high degree of overlapping of cache misses(and eventually memory accesses) makes it difficult to obtain an exact estimate of performance degradation. In response, we have developed a method that determines bounds on performance degradation by using carefully constructed synthetic benchmarks. Experimental validation with SPEC CPU2006 benchmarks shows that the observed performance degradation using these benchmarks is always within the predicted bounds. Further, a decrease of up to 43% is observed when comparing the predicted upper bound with the degradation predicted by a linear degradation model. Using the predicted degradation and the power profile of the system, which captures characteristics of the topology of the processors, it is possible to compute strict bounds on a metric based on performance/power for a power management strategy that uses DVFS. Our immediate future work concerns the extension of the performance degradation models shown in this paper to address other power management strategies, such as idle power management and those using server consolidation. The goal is to determine the best course of action in power management for each current system configuration and application. Towards this end, we will also evaluate our work with realistic, complex data center applications.

8. REFERENCES

[1] H. Amur, R. Nathuji, M. Ghosh, K. Schwan, and H.S. Lee. Idlepower: Application-Aware Management of Processor Idle States. In *Proceedings of MMCS, in conjunction with HPDC'08*, June 2008.

[2] L. Barroso and U. Hölzle. The Case for Energy Proportional Computing. *IEEE Computer*, 2007.

[3] F. Bellosa. The Benefits of Event-driven Energy Accounting in Power-sensitive Systems. In *Proceedings of the 9th ACM SIGOPS European Workshop*, 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. In *DATE04*, 2004.

[5] C. Hsu and U. Kremer. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. In *PLDI '03*. ACM Press.

[6] Intel. Dual-Core Intel Xeon Processor 5100 Series Datasheet. Technical report.

[7] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *MICRO 39*, 2006.

[8] R. Nathuji and K. Schwan. VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. In *SOSP '07*. ACM Press.

[9] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson. Application-Aware Power Management. In *IISWC'06*, pages 39–48, 2006.

[10] V. Srinivasan, G.R. Shenoy, S. Vaddagiri, D. Sarma, and V. Pallipadi. Energy-Aware Task and Interrupt Management in Linux. In *Proceedings of the Linux Symposium*, volume 2, August 2008.

[11] 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. In *MICRO 38*, 2005.

[12] F. Xie, M. Martonosi, and S. Malik. Bounds on Power Savings using Runtime Dynamic Voltage Scaling: An Exact Algorithm and a Linear-Time Heuristic Approximation. In *ISLPED'05*, 2005.