Multi-Optimization Power Management for Chip Multiprocessors

Ke Meng, Russ Joseph, Robert P. Dick
EECS Department
Northwestern University
Evanston, IL
k-meng@u.northwestern.edu,
{rjoseph,dickrp}@eecs.northwestern.edu

Li Shang
ECE Department
University of Colorado
Boulder, CO
li.shang@colorado.edu

ABSTRACT

The emergence of power as a first-class design constraint has fueled the proposal of a growing number of run-time power optimizations. Many of these optimizations trade-off power saving opportunity for a variable performance loss which depends on application characteristics and program phase. Furthermore, the potential benefits of these optimizations are sometimes non-additive, and it can be difficult to identify which combinations of these optimizations to apply. Trial-and-error approaches have been proposed to adaptively tune a processor. However, in a chip multiprocessor, the cost of individually configuring each core under a wide range of optimizations would be prohibitive under simple trial-and-error approaches.

In this work, we introduce an adaptive, multi-optimization power saving strategy for multi-core power management. Specifically, we solve the problem of meeting a global chip-wide power budget through run-time adaptation of highly configurable processor cores. Our approach applies analytic modeling to reduce exploration time and decrease the reliance on trial-and-error methods. We also introduce risk evaluation to balance the benefit of various power saving optimizations versus the potential performance loss. Overall, we find that our approach can significantly reduce processor power consumption compared to alternative optimization strategies.

Categories and Subject Descriptors

C.1.4 [**Processor Architecture**]: Parallel Architectures

General Terms

Design, Performance, Experimentation

Keywords

Chip Multi-Processor, Dynamic Power Management, Cache Resizing, Voltage/Frequency Scaling

1. INTRODUCTION

In modern systems, power optimizations at every layer of the system stack are crucial to avoid thermal limits and improve energy-efficiency. Both low-level optimizations at transistor scale and high-level operating system control of resources are indispensable in today's systems. An increasing number of optimizations at the micro-architecture level ([1, 6, 12]) seek to make run-time trade-offs with the goal of adapting to an application's needs. This task is difficult because not all of the optimizations have uniform power and performance effects on all applications. Furthermore, due to execution phases [10], it is likely that resource demands change frequently within an application's run and that consequently some power optimizations can become more or less desirable at various stages. The presence of multiple, compatible power optimizations, offers even greater potential for power-savings, but makes the decision process harder. Rather than merely a simple binary decision on whether or not to apply an optimization, the system must chose a subset of optimizations to apply. In some cases, these optimizations have non-additive effects, making it difficult to predict what the cumulative impact will be.

For multicore processors, the choice of which optimizations to apply is even more complicated because the global search space is potentially huge. In addition, multiprogrammed workloads offer challenges because each application is likely to have its own resource requirements. Furthermore, it can be difficult to balance these individual demands while trying to achieve global power and performance goals. Finally, a system-wide optimization is made harder by the presence of independent, disjoint program phases. Within each local application phase, we can expect a thread to maintain the same sensitivity to resource allocation and power optimizations. If we consider a system-wide phase to be an interval in which all threads are within stable phases, we can expect that on average these independent threads will contribute to make system-wide stable phases short lived. Consequently, global pressure on resources and hence susceptibility to power optimizations, evolves continually.

In recent work, trial-and-error frameworks have been used to manage reconfigurable units in a single core processor [9, 15]. In these approaches, the policy uses a testing period to try out all possible configurations or power modes. The results collected from the trial phase are further used to select an appropriate run-time configuration. To help reduce unnecessary trials, policies can include techniques [8] to identify stable execution phases. These execution windows are likely to have consistent resource utilization patterns. Trial phases are typically entered when the system detects that a new, stable execution phase has begun. If the phases are long-lived the system can benefit because the length of the trial phases are relatively short compared to the steady-state behavior.

This simple policy can be expanded to a multi-core scenario. However, certain features of this policy make it less favorable than it is in a single-core system. First, trial phases may contribute to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PACT'08, October 25–29, 2008, Toronto, Ontario, Canada.
Copyright 2008 ACM 978-1-60558-282-5/08/10 ...$5.00.

performance loss when inferior configurations are tested. Furthermore, the scalability of trial-and-error adaptation is quite limited. As more power-saving techniques and therefore more power modes are added, the number of required testing modes can explode.

In this paper, we examine a mechanism that integrates multiple power optimizations and globally manages chip multiprocessor processor power consumption to honor a chosen power budget. We believe this is timely given the prominence of multi-core processors and growing interest in run-time optimization. Our approach addresses concerns related to the large search space in a CMP system with many core-level optimizations, complex relationships between these optimizations, and transient resource demands due to very short-lived global phases.

Our work makes three principal contributions:

- We provide a global multiple optimization power management framework for multi-core architectures. Our solution configures individual cores to meet a chip-wide power constraint while maximizing global instruction throughput.
- Our approach introduces simple yet reasonably accurate performance/power models that allow us to explore a very large search space in a short amount of time. These models are *position independent*; they allow us to evaluate the power and performance of any given processor configuration from any other configuration.
- We propose a risk evaluation strategy to find a balance between cost and benefit of applying the various power-saving techniques.

The remainder of this paper is organized as follows: In Section 2, we describe our framework and identify how we envision global power management being used in a system. We go on to describe and evaluate power optimizations that we use in our solution in Section 3. Then, in Section 4 we discuss policies that can be used to guide global power management. We present simulation methodology, workloads, and experimental results in Sections 5 and 6. We follow with a discussion and comparison to related works in Section 7. Finally, we offer conclusions in Section 8.

2. SYSTEM FRAMEWORK

We begin with an overview of the proposed multiple-strategy multi-core power management system depicted in Figure 1. At design-time, we pre-select multiple power-saving strategies to form a candidate pool of optimizations. At run-time, we apply a global power manager to selectively configure the different optimizations (e.g., dynamic voltage frequency scaling, cache resizing) on individual processor cores to meet a specific power/performance goal. This same optimization target is also used in [16]. The power management system sits between the actual physical hardware and system-level software. We assume that the power manager is implemented in software. This is reasonable due to the flexibility and easier deployment of a software implementation. Power manager code can either be a part of the operating system or separately located in firmware.

The power manager chooses which optimizations to apply to each core given the workload and desired power/performance target. Each permutation of optimizations across cores is termed a *global power mode*. The number of possible power modes is dependent on the number of managed cores, the number of optimizations supported, and the degree to which optimizations are applied. Even if all of these quantities are modest, there can be a large number of possible global power modes. In this work, the power manager is charged with choosing a global power mode that obeys a processor-wide power budget while maximizing throughput [16].

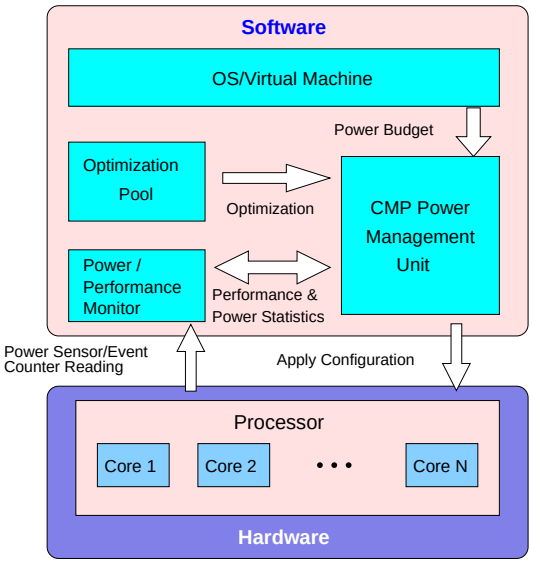
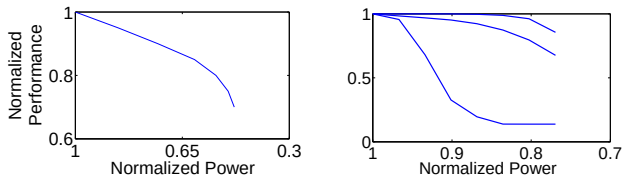


Figure 1: Global Multi-Strategy Power Management

The primary work of the global power management unit is to periodically evaluate the overall power usage and performance of the system and determine if it is warranted to switch to another power mode. The obvious scenarios where it may be prudent to change the power mode follow: (i) the power usage exceeds the global limit or (ii) the power management unit determines that another power mode offers better overall performance for the same power usage. The general goal is to maximize system throughput while satisfying a configurable power budget. This budget is not treated as a hard-limit the way that thermal constraints are [4, 32]. Indeed, we expect that temperature management facilities would also be present in the system. Our goal is to improve energy-efficiency and consequently, we treat the power budget as a soft-limit and make a best effort to stay under this ceiling. Because there is considerable uncertainty in program phases and power usage patterns [17], attempting to guarantee that a power budget is never exceeded might restrict the power manager to a small subset of power modes and hence hamper performance. We instead attempt to keep the power usage just below the budget while minimizing performance impacts.

Run-time monitoring is essential for both maintaining the power budget and delivering high instruction throughput; we apply a prediction scheme that combines core-level counters, sensors, and analytic models to keep tabs on the system in its current configuration and estimate how well other power modes might fare. For power monitoring, we assume that on-chip, core-level power measurements are available via circuit-level sensors. This functionality is present in recent industry processors [24]. We use a combination of performance event counters, cache miss counters, and analytic models to track performance. Our performance counters are inspired by CPI-stack style counters [11] which are now present in some high-performance processors.

Our global power management strategy applies an on-line performance/power modeling and optimization algorithm between intervals of fixed run-time. If well-designed, a CMP global power management policy can quickly detect changing system behavior, and react promptly to match the characteristics of the supporting hardwarde structures to the programs' new requirements.



(a) DVFS (b) Cache Resizing

Figure 2: Representative power-performance trade-offs

3. BUILDING A POWER OPTIMIZATION POOL

3.1 Candidate Pool

A large number of run-time power optimizations have been studied in recent years [6, 16, 8]. To construct a strategy pool with more optimizations, the power manager has more power modes to choose from and therefore can potentially achieve a better power savings if it could successfully select the best strategy each time. On the other hand, it could also become quite challenging and even hazardous when the manager has to evaluate, compare, and optimize over a large number of candidates. The decision on whether to include a particular power-saving strategy can both heavily affect and be affected by power management policy design. After a careful evaluation, we included two popular strategies in our study: Dynamic Voltage/Frequency Scaling (DVFS) and cache resizing.

DVFS is an efficient power-saving technique widely used in modern processors. In DVFS, dynamic power changes cubicly with frequency-voltage scaling [16] while static power changes exponentially with voltage [29]. And overall performance is roughly linear with clock frequency.

In this paper, we used the same power modes defined by Isci et al. [16]. The static power is approximated with a same cubic relationship as dynamic power since the results are very close within the voltage range. Table 1 lists all four allowed voltage-frequency pairs along with corresponding power-saving achieved and maximum performance loss. Notice that within this range there is a roughly 3-to-1 power-savings versus performance loss. Figure 2(a) shows a representative power-performance curve for DVFS.

Supply Voltage	Frequency	Power Saving	Max Perf-Loss
100%	100%	0	0
95%	95%	14%	5%
90%	90%	27%	10%
85%	85%	39%	15%

Table 1: Scaling effect of DVFS

In modern processors, caches occupy a large portion of die area and account for a large portion of the transistor budget. In Intel's dual-core 64-bit Xeon processor, more than 80% of the transistors are devoted to caches [33]. This results in a large contribution to static power usage from on-chip caches. As the feature size shrinks, leakage current continues to increase and static power becomes more significant. It is projected that static power will threaten the survival of CMOS itself [30] in the long term. Reducing cache leakage is a popular research topic and many cache resizing strategies have been proposed recently.

Both dynamic cache and static cache resizing attempt to adjust the cache size based on demand capacity. Figure 2(b) shows different power-performance trade-offs achieved on several workloads applying cache resizing. Different workloads and different stages within a workload have different tolerance to cache size reduction. Depending on the actual situation, cache resizing can provide an either better or worse result than DVFS as shown in the figure. While finer-granularity resizing is possible, in this work we apply a resizing technique with a granularity of cache ways, as in Albonesi's work [1]. When necessary, some ways of a cache will be turned off or put in a low-leakage state. Execution resumes with reduced cache capacity and associativity. We assume cache blocks that are turned off will stop consuming static and dynamic power. We apply cache resizing of the L1 instruction, L1 data, and unified L2 caches.

4. GLOBAL POWER MANAGEMENT POLICY

In this section, We detailedly discuss the design of our global multi-optimization power management policy. It solves the problem of maximizing overall instruction throughput on a homogeneous chip multiprocessor with a global power budget. Our approach is well suited to designs in which there are multiple power optimizations that can be applied to each processor core. Under simple trial-and-error approaches, there are an overwhelming number of global power modes. Even if the search space is pruned, it can still either lead to a long search process or force the power management system to make a poor choice.

The key to our approach is an analytic performance and power model that is fed by run-time counters and sensors. The model allows us to quickly evaluate a large number of power modes in far less time than it would take with trial-and-error evaluations. This is particularly important in multi-core systems since it may be desirable to re-evaluate power modes rather frequently (e.g., whenever any application encounters a phase change). We also introduce a form of risk evaluation that allows us to filter out power optimizations in which the potential negatives out-weigh the benefits.

4.1 Analytic Models

Analytic modeling for complex out-of-order processors has been recently proposed as a solution to design-time evaluation. In this work, we further extend analytic models to evaluate power optimizations at run-time.

A novel aspect of our approach is that our models are *position-independent*. Consequently, they allow us to evaluate the power and performance of any power mode from any other power mode, including the current one. The primary benefit of this is that we do not have to transition to a common or baseline configuration to evaluate an alternative set of power optimizations.

The projected performance loss and power savings for different power modes are calculated by feeding collected data from both sensors and event counters into simple yet reasonably accurate models. In this paper, we show how two power optimizations, namely dynamic voltage/frequency scaling (DVFS) and cache resizing via selective cache ways, can be evaluated on-line.

4.1.1 Modeling Dynamic Voltage/Frequency Scaling

As discussed in Section 3, DVFS is relatively easy to model and has rather predictable consequences on both power and performance. Overall, we can model power consumption to have a cubic relationship to the applied frequency. Given the power usage $Perf_i$ for a given application running at frequency $Freq_i$, the power consumption $Perf_j$ at an alternative frequency $Freq_j$ is given by:

$$Perf_j = Perf_i \times (\frac{Freq_j}{Freq_i})^3 \quad (1)$$

In previous work, on-line models have assumed a linear relation-

ship between frequency and performance [16]. This simple model is well suited to CPU-bound applications in which there are relatively few interactions with the memory system. Because globally shared caches and off-chip RAM are outside the scope of core-level DVFS, stall time in the memory does not scale with frequency. Memory-bound applications are therefore less impacted by frequency scaling. Our run-time model can explicitly quantify this effect by estimating the division between compute and memory time.

Our approach uses CPI stack counters [11], to measure the fraction of execution time that is spent on memory system access and models that take into account the asymmetric effects of frequency scaling. Performance event counter architectures that classify execution cycles have gained interest recently [8, 11], and have been featured in commercially available processors [25]. They are useful in profiling, performance debugging, and system analysis because they provide detailed breakdowns that would otherwise only be obtainable through simulation or extremely high-level estimation. They can directly measure execution stalls due to cache misses in many levels in the hierarchy, TLB misses, branch mispredictions, and pipeline resource contention. In this work, we re-purpose them to collect the division between compute and memory stalls and hence focus only on two types of counts: total cycles and L2 miss stalls (instruction + data). If we consider CPU and Mem to be compute and memory stall times, we can model performance at two frequencies $Freq_i$ and $Freq_j$ as

$$CPU_j = CPU_i \times (\frac{Freq_j}{Freq_i}) \tag{2}$$

$$Mem_j = Mem_i \tag{3}$$

The projected new performance can be calculated as

$$Perf_j = \frac{CPU_i \times (\frac{Freq_j}{Freq_i}) + Mem_i}{CPU_i + Mem_i} \times Perf_i \tag{4}$$

4.1.2 Modeling Cache Resizing

Our performance model for cache resizing depends on two separate components. First, we model the CPI cycle penalty of an individual cache miss. Second, we approximate how the current miss rate would change at an alternative cache size. The product of these terms is the expected performance benefit of applying cache resize.

The instantaneous per miss cycle penalty (e.g., the average cost of a cache miss under the current cache configuration) is computed individually for caches at each level of the memory hierarchy. We identify CPI cycles associated with cache misses using the CPI stack model proposed by Eyerman et al. [11]. We then divide those miss cycles by the number of misses in that cache. This yields the associated miss penalty. We assume that miss penalty is an application-dependent characteristic and it is independent of miss rate. This is not strictly true because misses often overlap. Consequently, as the miss rate increases, the average miss penalty may decrease if the miss events are clustered. Nevertheless, we find that the simple miss penalty calculation is sufficient because it is rarely prudent to select a cache configuration that increases the miss rate significantly. It can be considered a conservative estimate.

To predict the miss rate for an alternative cache size, we apply the method used by Qureshi and Patt [28], which itself exploits the cache's stack property[23] under the LRU replacement policy. A hit at a smaller cache size is guaranteed to also be a hit when the associativity increases. Figure 3 shows the extra counters needed to book-keep hit and miss events. The cache tag is logically organized in MRU to LRU order and a hit counter is connected to each column of tags that share the same ranking in the recent access order. A hit in the tag will result an increment in corresponding counter and move the tag to the MRU position. Any misses in the cache tag will be recorded in a separate miss counter. The presumed change of the number of misses between different cache associativity can be observed using the stats collected from all these counters. As the example in Figure 3 shows, if the cache is currently sized to be two-way set associative, the number of cache misses will be reduced by 23 if the associativity increases to three-way. This is true because those hits counted in third most recently visited tags would become true hits with increased capacity and associativity.

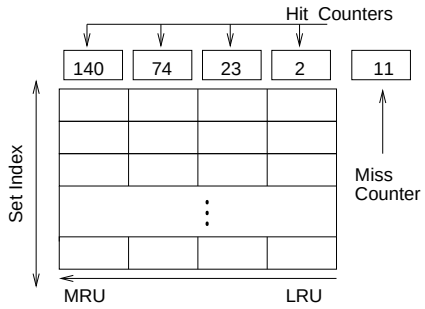


Figure 3: Supplementary counters for cache miss statistics.

We can now model the performance for a different associative configuration using current and predicted miss rates, base CPI (non-stall) cycles and stall cycles due misses in this cache. Assume the current miss rate is $MissRate_i$, the current miss penalty is $MissPenalty_i$ and the base CPI is represented by $BaseCPI$. Consider a cache that is resized to an alternative associativity and capacity with projected miss rate $MissRate_j$. We can estimate the performance impact as follows:

$$MissPenalty_j = MissPenalty_i \times (\frac{MissRate_j}{MissRate_i}) \tag{5}$$

$$Perf_j = \frac{MissCPI_j + BaseCPI}{MissCPIi + BaseCPI} \times Perf_i \tag{6}$$

Finally, we consider power models for resized caches. We assume that we can make use of either direct cache leakage sensors [20] or that we can model static power using thermal sensors to detect local temperature and coupling those readings to cache leakage models [15]. Either approach allows us to estimate the leakage power of the cache. We then assume that the static power usage of a cache is then equal to the average leakage power of a cache way times the number of enabled ways.

4.1.3 Unified Analytic Models

In general, the power optimizations we describe are not strictly independent. For example, performance under DVFS depends on the ratio of memory stall cycles to compute cycles. Compute-bound applications have few memory stalls and hence are directly affected by any decrease in clock frequency. An optimization like downsizing the L2 cache may increase miss rate and hence memory stall cycles. This could potentially decrease sensitivity to DVFS, and hence make frequency scaling effects less acute for compute-bound applications.

In practice, we find that the interactions between the power optimizations that we evaluate to be rather small. Consequently, we do not directly include these dependencies in our models. This greatly simplifies our run-time projections and selection algorithms. In our case, the interactions between power optimizations are only visible in extreme cases. For example, the dependence between cache resizing and frequency scaling is mostly visible when the miss rate makes an abrupt transition, causing the memory stall cycles to jump

similarly. Because any significant increase in miss rate is likely to have an extremely negative impact on performance, the power manager would likely avoid it. Even though there might be some error in projecting the performance of simultaneously resizing the cache and varying the frequency, the model would lead us to draw the right qualitative conclusion: namely that this is a poor choice.

Treating each candidate power optimization as an independent factor allows us to compute simple speedup factors. We can either examine the relative impact of an optimization in isolation or view the aggregate impact of a set of different optimizations by computing the product of their speedup factors. Given a set of optimizations, each having a speedup $OptSpeedup_k$, we can mathematically represent these computations as

$$\frac{Perf_j}{Perf_i} = \prod_k OptSpeedup_k \tag{7}$$

Power modeling is slightly complicated by the fact that cache resizing has a simple additive impact on power while DVFS has a strong, super-linear influence on total core power. We model the additive contributions of cache resizing by summing up a set of additive power delta values ($\Delta OptPower_1, OptPower_2, \ldots$) before multiplying the frequency scaling effect between frequencies $Freq_i$ and $Freq_j$. Notice the additive power delta values are the reduced or increased leakage power under current voltage-frequency pair due to the adjustment of cache size. The formula for power modeling follows:

$$Power_j = (Power_i + \sum_k \Delta OptPower_k) \times (\frac{Freq_j}{Freq_i})^3$$

4.2 Risk Evaluation in Analytic Modeling

Compared to trial-and-error solutions, run-time management that is guided by an analytic model can offer a significant reduction in search time. This may allow the power manager to find an adequate power mode early in the execution phase, maximizing power and performance benefits. However, we acknowledge that this style of power management can introduce some uncertainty. In particular, we do not reconfigure the system while exploring the search space. This opens the possibility of skipping a potentially-beneficial configuration that would not have been missed under a more conservative trial-and-error based search.

In our evaluation, we found two scenarios where this posed a problem. In the first case, we inadvertently select a power mode because we presume that the current execution phase is stable. If this is not true – the phase is short lived – we could encounter a potentially-harsh reconfiguration penalty and be temporarily stuck in a less-than-ideal configuration mode. This is sometimes true in cases when cache demands drop momentarily, tricking the power manager into downsizing the cache, only to have demand capacity rise sharply thereafter. For a trial-and-error power manager, then end of a stable phase generally leads to a back-off interval during which power optimizations are disabled until the phase has stabilized. Search space exploration begins anew once a stable phase has been detected. Consequently, short bursts of instability are less likely to lead to dramatic performance losses.

In the second case, the power manager chooses a poor configuration due to modeling error. This could have several outcomes. First, if the actual performance is worse than the predicted performance and phases remain stable, then we could potentially be stuck in an inferior power mode for a long time. Second, if the predicted power is significantly higher than expected, we might temporarily exceed the power budget. This violation would be detected and the power manager would try to scale the power to bring the system into compliance. If, however, this modeling error was persistent, it would be possible to once again transition to a high-power mode, repeating the cycle. We note that in all of our studies we found the analytic models had enough fidelity that these behaviors were extremely rare.

A compromise can be made if we realize that we have several power optimizations to choose from and these strategies possess various stabilities, as shown in Figure 2. Unlike cache resizing, DVFS can provide more predictable power-performance trade-offs for many workloads. This means when given a choice of multiple optimizations it may be wiser to temporarily ignore an optimization if we are not confident in our power and performance predictions for it. In this scenario, we lean more heavily on optimizations for which we have more confidence in predictions and/or stability. The potential price for this pruning is a optimal trade-off of power and performance. Nevertheless, we can reduce the chances of performance loss caused by errant predictions and still achieve a decent power savings.

Risk functions can be developed for each power optimization in the pool. These functions track previous history values for our event counters or sensors and determine if resource utilization patterns are consistent enough to apply projections and make use of optimizations. If the risk function indicates that confidence for a given optimization is low, we can choose to temporarily disable it.

We propose a simple set of risk evaluation functions that help to reduce exposure to performance loss. We assume that DVFS offers extremely predictable power and performance and for the purposes of our studies assume that it has zero risk. For cache resizing we use a simple risk function based on windowed history of power usage. For each evaluation interval, we record core-level power consumed in the current power mode and normalize it by the projected power consumption in the maximal config mode. We calculate the standard deviation of this normalized power consumption over a range of previous evaluation intervals. This captures how consistently an application stresses the power budget relative to its performance. The idea behind this risk function is that when cache behavior is not stable, it is most likely that the program is bouncing between CPU-bound and memory-bound patterns, and the power usage also will vary because memory-bound applications normally use less power. We found that a history of eight to ten intervals worked well in practice. We chose acceptance thresholds of 0.4, 0.4, and 0.5 for the L1-instruction, L1-data, and L2 caches. When the standard-deviation exceeds these thresholds, we disable resizing strategies for those caches and reconfigure to use the maximum capacity.

4.3 Search

To find the best power modes, one option is to completely search through all local power modes to determine which subset of optimizations to apply. Here we can only choose one power mode for each processor core and this forms a multiple-choice knapsack problem [2], which is NP-hard. The search space can quickly explode when more cores, more power-saving strategies, and therefore more power modes are considered. Greedy algorithms can sometimes be used to find near-optimal solutions to some knapsack problems. We find that for our problem, greedy search offers high-quality solutions in a relatively few steps.

In each evaluation period, our greedy algorithm begins by considering the power and performance of the maximal power mode (e.g., one in which all power optimizations are disabled for every core). It determines whether or not this configuration exceeds the power budget. If it does not, the search concludes. Otherwise, the algorithm calculates the power and performance of every immedi-

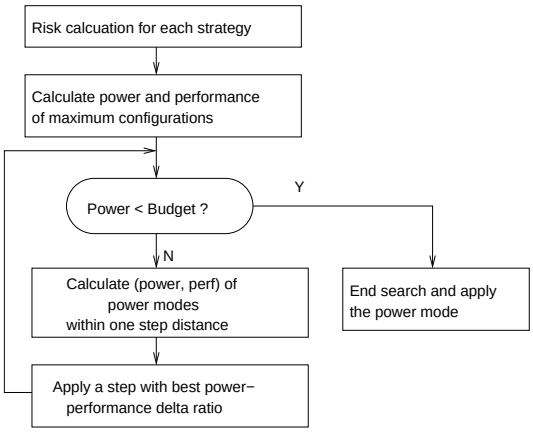


Figure 4: On-line evaluation algorithm.

ately neighboring downgraded global power mode. It then selects the power mode with the maximal delta power/performance ratio. This corresponds to the global configuration that offers the steepest reduction in power. We repeatedly evaluate until we meet the power budget. At this point, the algorithm terminates and the power manager applies the current configuration. Figure 4 presents a flowchart that summarizes our algorithm.

We begin each application of the search with the same start node (the maximal power mode). We could instead have initialized the search with the current CMP configuration. This seems like an alternative that would offer an extremely quick search assuming that there are relatively few power mode transitions. In the common case, the power mode used in the last interval will be repeated in the subsequent interval. However, this approach can easily get trapped in a local optimum and achieve mediocre results. By anchoring the initial step of the search process to the maximal config, we essentially avoided local minimal.

4.4 Overhead

Large overhead for any power-saving strategy should be avoided since the additional power usage from extra hardware or software required can waste or even consume much of the power savings achieved. Careful consideration must be taken when designing a multiple-optimization power management scheme. We believe that our design, maintains low overhead. We require separate phase-locking circuits for each core and some event counters. Commercially available processors commonly feature an impressive array of event counters and support core-level voltage/frequency scaling [25]. Our approach includes additional software overhead in the form of calculation of power-performance estimates for different power modes, risk evaluations for every strategy, and a greedy search through the space. We assume that our evaluation intervals are $600\mu s$ or longer. We expect that the overall impact on execution time is immaterial. In general, an efficient implementation of the design bears negligible overhead compared to the power savings achieved.

5. EXPERIMENTAL METHODOLOGY

In this section, we describe the multi-core processor model and workloads used in this paper.

5.1 Processor Model

Our experiments model a 4-core homogenous chip multiprocessors for a 65 nm process. Each core of the processor is comparable to an Alpha 21264 (EV6) scaled to current technology [14]. A similar scaling methodology was used by Kumar et al. in [21]. We assume that each processor core runs with a maximum frequency of 3.0 GHz. The cores in the processor have private L1 data and instruction caches and private L2 unified caches. Inter-core communication and off-chip memory transfers travel across an on-chip bus network. Table 2 summarizes our base processor model.

Single Core	
Max Clock Rate Allowed	3.0 GHz
Fetch/Decode Width	4 inst
Issue Width	6 inst, out-of-order
IQ/LSQ/ROB	32/40/80 entries
Functional Units	4 IntALU, 1 IntMult/Div 1 FPALU, 1 FPMul/Div 2 MemPorts
L1 Inst Cache	64KB 2-way 64B blocks
L1 Data Cache	64KB 2-way 64B blocks 3 cycle load hit
Chip Multiprocessor	
Cores	4
L2	2MB 16-way private 128B blocks
Max Off-chip memory latency	250 cycles
Power Parameters	
Max VDD	1.0V
Max Clock Rate	3.0GHz
Feature Size	65nm

Table 2: Processor Parameters

We assume that each core maintains its own clock domain and that processor frequencies can be scaled independently to a discrete set of operating frequencies. The discrete frequency points include 85%, 90%, 95%, and 100% of the maximum frequency. The corresponding voltage supplies for these frequency settings are also 85%, 90%, 95%, and 100% of the nominal power voltage required for the maximum frequency.

5.2 Simulation Framework

The simulation infrastructure uses a heavily-modified version of the M5 cycle-accurate Stand-Alone Execution simulator [3], which includes detailed models of pipelines, caches, buses, and off-chip memory. We have modified M5 to support the simulation of DVFS at core-granularity and dynamic cache resizing at way-granularity.

To simulate power usage, we merged dynamic power models from Wattch [5] into M5. Wattch is a widely-used architecture-level power simulator and provides a reasonably-accurate dynamic power model. To simulate static power, we adapt leakage data from a commercial processor [26]. This data was collected from a high-speed thermal scope, which measured localized component temperature in a processor. The temperature readings were then analyzed to identify dynamic and static power breakdowns on a structural basis. Using the static power densities as a baseline, we used scaling factors from ITRS [30] to calculate static power density at 65 nm. Finally, we multiply the static power densities by the component areas using a floorplan of an EV6 core scaled to 65 nm. On average, the static power is around 40% of the total power usage, which is on par with the ITRS figures [30]. We simulate any necessary penalty cycles needed for data write-back during a cache sizing-down and a fixed number of penalty cycles for frequency scaling.

5.3 Workloads

To better evaluate different scenarios, we create four workload groups that showcase representative processor usage patterns. Each group consists of four applications taken from the SPEC CPU2000 benchmark suite. To isolate representative simulation windows, we

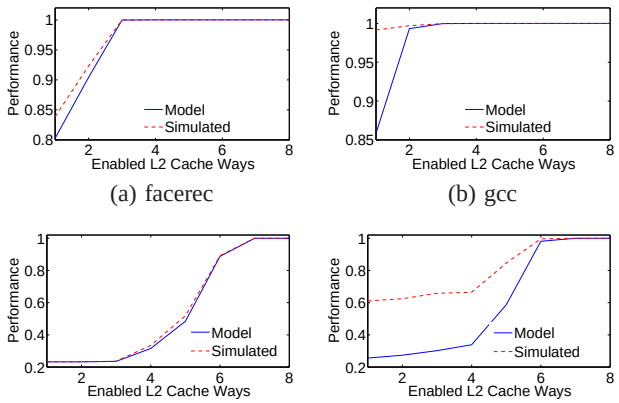


Figure 5: Validating the L2 CPI Model

use SimPoint [31] to identify relevant instruction execution intervals for all benchmarks and save checkpoints. Using these checkpoints, we simulate until the slowest application has run 300 million instructions. Table 3 lists the actual workload compositions. The workloads for group C and group D are specially selected. All applications in group C show relatively-stable program phases while the opposite is true for group D. The remaining groups consist of randomly picked programs.

Groups No.	Workloads
Group A	equake, swim, sixtrack, gcc
Group B	applu, gap, facerec, vortex
Group C	mesa, eon, lucas, wupwise
Group D	art, mcf, parser, vpr

Table 3: Workloads

6. EXPERIMENTAL RESULTS

6.1 Validation Cache CPI Model

The accuracy of our analytic models is important since the whole power management system is built upon it. Any serious modeling errors can easily cause inferior power mode adjustment decisions. We found that our DVFS models are quite accurate given the relatively-simple relationship between performance, frequency, and CPU versus memory stall cycles. The deviation is within 4%. We focus our discussion on the verification of the CPI modeling needed for cache resizing.

We first ran a batch of simulations for SPEC 2000 benchmarks with a full-size L2 cache and used our analytic CPI models to calculate the expected performance at various resized cache capacities. In a second round of simulations, we statically resized the cache to each of the sizes and simulated actual performance. We show the results for four representative benchmarks in Figure 5. The solid lines in these graphs show the normalized performance calculated from CPI models and the dashed lines show the results when we actually resize the cache. Clearly, the model is very accurate for both facerec and mcf. There is small disagreement between the model and simulation when the cache size becomes very small in Figure 5(a). Sometimes the distance can be very large as shown in the case of gcc. This is because when the cache size becomes minimal, cache miss rates can increase significantly. With more cache misses, the average cache miss penalty will drop because

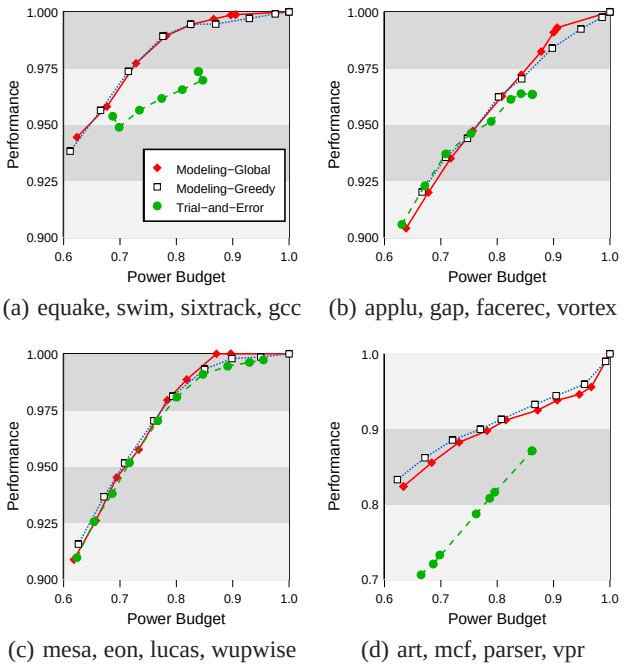


Figure 6: Comparing trial-and-error search space evaluation with model directed search.

more misses can overlap. A pessimistic penalty estimate based on larger cache sizes can cause conservative performance estimates. Most of the time, this happens when the distance between actual power mode and the predicted power mode is large (e.g., modeling the performance for minimal-sized cache from maximal-sized cache in our case). Overall, the model works very well except for art, which has an extremely high miss rate and many overlapping memory accesses. Consequently, we overestimate the miss penalty. However, this is not a severe problem for two reasons. First, this kind of behavior is relatively rare. Other than art, mgrid is the only other benchmark that shows a similar behavior. Second, the predicted performance is generally pessimistic. This means that we are unlikely to make an ill-advised cache size reduction in a power manager.

6.2 Model Evaluation versus Trial-and-Error

In order to provide a comparison, we also implemented a trial-and-error power management policy. A trial interval for a workload will begin when the program ends an unstable period and enters a new stable series of phases. To detect the phase change, a detection method based on code-signature [10] is also implemented. The signature is a 1,024-bit value xor-ed from hash values based on addresses of executed instructions. A distance value between two signatures is calculated and a phase change is thought to occur when the distance is larger than an empirically-determined threshold. For the global trial-and-error power management policy, when one or several workloads are unstable, maximal config will be applied to the cores holding those benchmarks and adjustments will be made to the remaining cores to meet the power budget.

In Figure 6, the curves labeled *Modeling-Greedy* shows the performance degradation under a given power budget for our analytic-modeling method using greedy search but without risk evaluation. The curves labeled *Trial-and-error* represent the results of applying the trial-and-error management policy. Table 4 lists the the time

percentage of unstable stages for both single workload and the processor as a whole structure. The only situation in which the trial-and-error policy works well is for Group C, which also shows high stability. For Group B, even though only one workload is highly unstable, globally the whole system is impacted and it is difficult to achieve a global balance using the trial-and-error policy. Also, it is very difficult for the trial-and-error policy to meet an exact power usage target. On the contrary, our analytic-modeling-based policy without any risk evaluation strategy can easily achieve decent power-performance trade-offs and on average, a 8% performance degradation can save 35% power.

Group A.	Unstable Period	Group B.	Unstable Period
equake	12.7%	applu	5.2%
swim	20.9%	gap	1.5%
sixtrack	0%	facerec	52.4%
gcc	43.6%	vortex	0.7%
Processor	61.8%	Processor	55%
Group C.	**Unstable Period**	**Group D.**	**Unstable Period**
mesa	2.3%	art	96.2%
eon	0%	mcf	49.7%
lucas	17.4%	parser	44.8%
wupwise	0%	vpr	18.5%
Processor	19.8%	Processor	99.1%

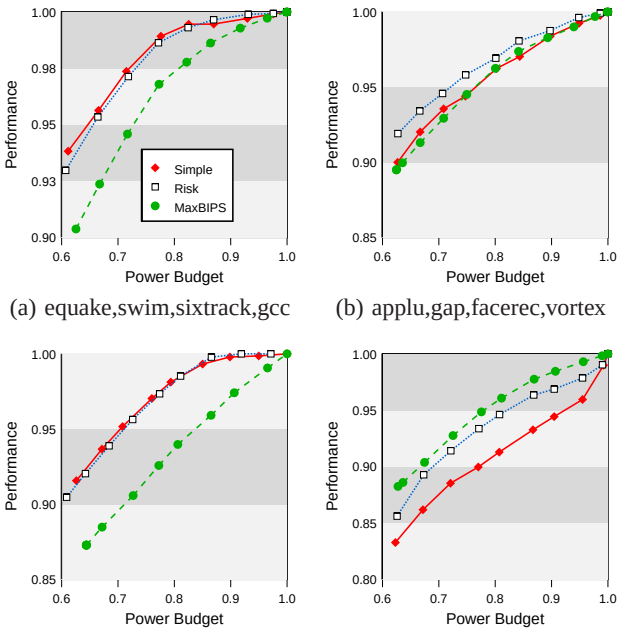
Table 4: Percentage of time that individual threads exhibit unstable phase behavior.

In Figure 6, we also compare the results for modeling-based methods with a greedy heuristic and a global brute-force optimal search. The curve labeled *Modeling-Global* shows the results when an optimal search among all combinations of power modes is performed. It is clear that the results for a greedy search and a globally-optimal search are extremely close. For some cases, the optimal search actually provides a slightly larger performance loss for same amount of power savings. This is because in an optimal search, more configuration changes are made given a presumably-small benefit, which is not enough to compensate for the performance loss during a configuration change, such as forced write-back of valid content when sizing-down a cache.

6.3 Modeling with Risk Management and MaxBIPS

Figure 7 shows another group of results for the four workload groups selected. In these figures, the curves labeled *Modeling* still represent the power management results for the policy based on analytic modeling without a risk evaluation support. The curves labeled *Modeling-Risk* show power-performance trade-offs when the risk evaluation strategy has been applied. Finally, the curves labeled *MaxBIPS* show the results of a global power management policy introduced by Isci et al. [16]. MaxBIPS applies DVFS with core-level granularity based on very simple analytic models that do not consider the ratio between memory and compute cycle. Isci et al. [16] demonstrated that MaxBIPS can achieve results that are close to an oracle situation, in which future knowledge is used to direct current configuration adjustment. This means further improvement on solely DVFS-based strategy would be extremely difficult to obtain.

For all four groups of workloads, our policy with simple risk evaluation support provides either an equivalent or better results depending on the stability of the workloads. The best improvement happens with the group composed of four unstable workloads as shown in Figure 7(d). These results indicate that a simple cache-resizing risk evaluation strategy based only on power usage variation can work well in practice. It manages to avoid inferior configurations that would adversely impact performance while not missing out on good opportunities to save power. It is worth to point out that the workload group composed of equake, swim, sixtrack, and gcc is not friendly to a trial-and-error policy because of frequent phase changes as identified by code signatures. However, the actual cache usage is quite stable as shown in Figure 7(a). Our risk management policy notices the stable cache behavior and allows user to apply resizing to achieve good power savings.



(a) equake,swim,sixtrack,gcc (b) applu,gap,facerec,vortex

(c) mesa,eon,lucas,wupwise (d) art,mcf,parser,vpr

Figure 7: Modeling with Risk Management and MaxBIPS

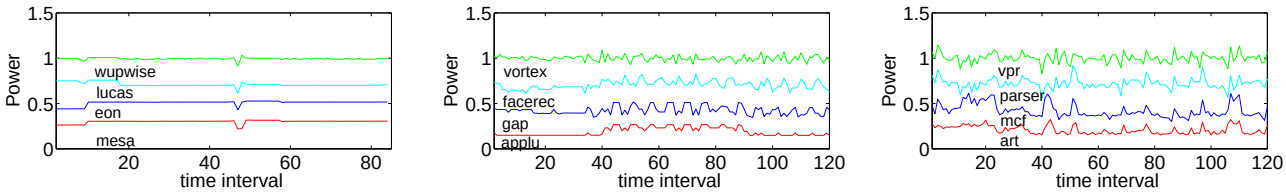
When compared to MaxBIPS, multi-optimization based power management policy does not always win out. In Figure 7(b), MaxBIPS and the policy without risk evaluation support have very close results. In Figure 7(d), MaxBIPS beats both of our policies. This illustrates the difficulty of managing multiple strategies and designing an efficient policy. Even though risk evaluation can be efficient in helping identify the strategies that temporarily possess high risks and are likely to hurt performance, it often takes time to notice the risk (several evaluation intervals). By that time, some errors may have already been made.

6.4 Power Violations Over Time

In this work, we are more interested in achieving an average power usage target. Therefore the power budget is not treated as a hard constraint and temporary budget violations are allowed. In this section, we examine the frequency of budget violations.

In this work, we are primarily interested in achieving an steady-state power usage target. We presume that the hardware has built-in thermal throttling mechanisms that engage whenever a core approaches a potentially-dangerous temperature peak. Consequently, we do not treat the power budget as a hard constraint in our work. Occasionally, budget violations may occur, and when they do, the global power manager detects them and transitions the system to a lower power state. In this section, we examine the frequency of budget violations and their relationship to application characteristics.

Figure 8 shows a normalized power break-down for a set of



(a) mesa,eon,lucas,wupwise (b) applu,gap,facerec,vortex (c) art,mcf,parser,vpr

Figure 8: Power usage break-down

workloads under a given power budget. From the graph, power usage is very stable for a workload group that consists of highly-predictable benchmarks as in Figure 8(a). In the case of Figure 8(b) and 8(c), fluctuations in power usage are more prominent and temporary power budget violations happen more frequently. In particular, for the workload consisting of art, mcf, parser, and vpr the normalized power budget of one is exceeded many times. This plot is consistent with results on phase stability from the previous sections. These applications have rather unstable resource usage, and it is difficult to predict their power or performance characteristics. This leads to repeated violations.

We believe that further improvements in global management can be made by introducing a safety-margin just below the global power budget. The power manager would conservatively seek to maintain this set-point, giving us enough breathing room to decrease the number of true violations. Furthermore, it may be possible to adaptively determine an ideal value for this set-point by monitoring previous behavior, and in particular, tracking true or near budget violations.

7. DISCUSSION AND RELATED WORKS

Architecture-level power management has long been an important topic given that both high power usage and heat generation in high-performance processors act as major design obstacles and challenge further technology scaling. Many thermal and power-saving techniques have been proposed recently [4, 6, 13].

Some of these techniques target a particular architectural unit and others take a more global aim at power. Cases where these optimizations do not completely overlap can offer great potential. Specifically, Huang developed a framework that was capable of achieving very good processor-wide energy-savings by combining several different power optimizations [15]. This approach pre-sorts several techniques based on their general efficiencies and applies them one by one until a pre-set limit is satisfied. While this greatly simplifies the design of the power manager, it does not make allowances for cases where the relatively efficiency of the power-saving approaches change due to application characteristics or context.

Work by Dhodapkar and Smith examined the concept of working-set based optimization [8, 9]. In particular, they developed an on-line phase detection approach and showed how it can be coupled with trial-and-error to efficiently tune processor resources to save power. In his thesis, Dhodapkar suggests that it might be possible to reduce or eliminate the dependence on trial-and-error through compact analytic models [7]. In later work, Karkhanis and Smith introduced analytic models that could be used to evaluate out-of-order processors at design time [18, 19]. In this work, we apply Dhodapkar's notion of using analytic models on-line to predict performance under a number of complex configurations.

Power optimizations play an increasingly-important role in multi-core architectures. Recent work has examined benefits of heterogeneous architectures for power efficiency [21], power and temperature constraints for multi-core systems[22], and OS-level thermal management for CMPs [27].

Isci et al. were the first to propose the concept of a global chip-level power budget for a multi-core processor [16]. This work also introduced the notion of a global power management unit and proposed several CMP global power management policies using only DVFS techniques. In particular, the MAXBIPS policy demonstrates impressive results by applying a set of very simple analytic models to predict performance and power under DVFS.

We extend this work by integrating several power optimizations together and designing an efficient, yet light-weight global power management policy. From here, we believe further exploration is necessary. While more power saving strategies can be integrated into the pool, improvement on the policy can also be profitable. We plan to investigate a wider range of power optimizations and tighter control on global power in our future work.

8. CONCLUSION

We described a global multiple optimization power management framework for multi-core architectures. Our approach applies a greedy algorithm to quickly examine a large search-space and find operating points that offer good power/performance compromises. We then configure individual cores to meet the chip-wide power constraint while maximizing global instruction throughput. Our approach introduces simple yet reasonably-accurate power models that allow us to explore a very large search space in a short amount of time. In particular, we introduce position-independent models that allow us to evaluate potential power modes based on the current configuration. With these mechanisms, we find that we can avoid the cost of trial-and-error approaches commonly used in on-line microarchitecture adaptation. Finally, we propose a risk evaluation strategy to find a balance between cost and benefit of applying the various power-saving techniques.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. This work was supported in part by NSF CAREER Award CCF-0644332 and Awards CCF-0541337, CNS-0720820 and CCF-0702761, and in part by SRC award 2007-HS-1593.

9. REFERENCES

[1] D. H. Albonesi. Selective cache ways: On-demand cache resource allocation. In *International Symposium on Microarchitecture*, Nov., 1999.

[2] R. D. Armstrong, D. S. Kung, P. Sinha, and A. A. Zoltners. A computational study of a multiple-choice knapsack algorithm. *ACM Transactions on Mathematical Software*, 9(2), June 1983.

[3] N. L. Binkert, E. G. Hallnor, and S. K. Reinhardt. Network-oriented full-system simulation using M5. In *Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, February 2003.

[4] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA-7)*, January 2001.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture (ISCA-27)*, June 2000.

[6] A. Buyuktosunoglu, T. Karkhanis, D. H. Albonesi, and P. Bose. Energy efficient co-adaptive instruction fetch and issue. In *Proceedings of the 30th International Symposium on Computer Architecture (ISCA-30)*, May 2003.

[7] A. Dhodapkar. *Autonomic Management of Adaptive Microarchitectures*. PhD thesis, University of Wisconsin-Madison, 2004.

[8] A. Dhodapkar and J. E. Smith. Managing multi-configuration hardware via dynamic working set analysis. In *Proceedings of the 29th International Symposium on Computer Architecture (ISCA-29)*, May 2002.

[9] A. Dhodapkar and J. E. Smith. Tuning recongurable microarchitectures for power efficiency. In *The 11th Recongurable Architectures Workshop (RAW 2004), held in conjunction with the 18th International Parallel and Distributed Processing Symposium*, April 2004.

[10] A. S. Dhodapkar and J. E. Smith. Comparing program phase detection techniques. In *MICRO-36*, December 2003.

[11] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith. A performance counter architecture for computing accurate CPI components. In *ASPLOS'00*, December 2000.

[12] D. Folegnani and A. Gonzalez. Reducing power consumption of the issue logic. In *Proceedings of the 28th International Symposium on Computer Architecture (ISCA-28)*, July 2001.

[13] R. Gonzalez and M. Horowitz. Energy Dissipation in General Purpose Microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–84, 1996.

[14] L. Gwennap. Digital 21264 sets new standard. *Microprocessor Report*, pages 11–16, Oct. 28, 1996.

[15] M. Huang, J. Renau, S.-M. Yoo, and J. Torrellas. A framework for dynamic energy efficiency and temperature management. In *MICRO-33*, December 2000.

[16] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 36th International Symposium on Microarchitecture (MICRO-39)*, December 2006.

[17] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th International Symposium on Microarchitecture (MICRO-36)*, December 2003.

[18] T. Karkhanis and J. E. Smith. A first-order superscalar processor model. In *ISCA-31*, June 2004.

[19] T. Karkhanis and J. E. Smith. Automated design of application-specific superscalar processors. In *ISCA-33*, June 2006.

[20] S. Kaxiras and P. Xekalakis. 4t-Decay sensors: a new class of small, fast, robust, and low-power, temperature/leakage sensors. In *ISLPED' 04*, August 2004.

[21] R. Kumar, K. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the 36th International Symposium on Microarchitecture (MICRO-36)*, December 2003.

[22] Y. Li, K. Skadron, Z. Hu, and D. Brooks. Performance, energy, and thermal considerations for SMT and CMP architectures. In *Eleventh International Conference on High Performance Computer Architectures (HPCA-11)*, February 2005.

[23] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Journal of Research and Development*, 9(2), 1970.

[24] R. McGowen, C. A. Poirier, C. Bostak, et al. Power and temperature control on a 90-nm Itanium Family processor. *IEEE Journal of Solid-State Circuits*, 41(1), January 2006.

[25] A. Mericas. Performance monitoring on the Power5 microprocessor. In *Performance Evaluation and Benchmarking*, pages 247–266. CRC Press, 2006.

[26] F. J. Mesa-Martinez, J. Nayfach-Battilana, and J. Renau. Power model validation through thermal measurements. In *ISCA-34*, June 2007.

[27] M. Powell, M. Gomaa, and T. Vijaykumar. Heat-and-run: Leveraging SMT and CMP to manage power density through the operating system. In *Proceedings of the Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XI)*, October 2004.

[28] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Micro-39*, December 2006.

[29] R. Rao, A. Srivastava, D. Blaauw, and D. Sylvester. Statistical analysis of subthreshold leakage current for VLSI circuits. *IEEE Trans. on VLSI Systems*, 12:131–139, Feb. 2004.

[30] Semiconductor Industry Association. International Technology Roadmap for Semiconductors, 2007. http://www.itrs.net/Links/2007ITRS/Home2007.htm.

[31] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct 2002.

[32] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th International Symposium on Computer Architecture (ISCA-30)*, June 2003.

[33] S. Tam, S. Rusu, J. Chang, S. Vora, B. Cherkauer, and D. Ayers. A 65nm 95W Dual-core multi-threaded Xeon Processor with L3 cache. In *IEEE Asian Solid-State Circuits Conference*, November 2006.