Achieving Fair or Differentiated Cache Sharing in Power-Constrained Chip Multiprocessors

Xiaorui Wang, Kai Ma, and Yefu Wang
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN 37996
{*xwang, kma3, ywang38*}*@eecs.utk.edu*

Abstract

Limiting the peak power consumption of chip multiprocessors (CMPs) has recently received a lot of attention. In order to enable chip-level power capping, the peak power consumption of on-chip L2 caches in a CMP often needs to be constrained by dynamically transitioning selected cache banks into low-power modes. However, dynamic cache resizing for power capping may cause undesired long cache access latencies, and even thread starving and thrashing, for the applications running on the CMP. In this paper, we propose a novel cache management strategy that can limit the peak power consumption of L2 caches and provide fairness guarantees, such that the cache access latencies of the application threads co-scheduled on the CMP are impacted more uniformly. Our strategy is also extended to provide differentiated cache latency guarantees that can help the OS to enforce the desired thread priorities at the architectural level and achieve desired rates of thread progress for co-scheduled applications. Our solution features a two-tier control architecture rigorously designed based on advanced feedback control theory for guaranteed control accuracy and system stability. Extensive experimental results demonstrate that our solution can achieve the desired cache power capping, fair or differentiated cache sharing, and power-performance tradeoffs for many applications.

1 Introduction

In recent years, limiting the peak power consumption of chip multiprocessors (CMPs) has received a lot of attention [11, 27, 16, 30]. With the increased levels of core integration and transistor density, the gap between the peak and average power consumption of a CMP widens and leads to unnecessarily more expensive cooling, packaging, and power supplies in the CMP design. To effectively reduce the costs while allowing higher computing densities with more cores and caches integrated on a single die, power capping can be enforced at different levels of a CMP for maximized performance under given power constraints. In addition, to enable chip-level power capping, it is preferable that power can be flexibly shifted among the CPU cores and the shared on-chip L2 caches within a CMP for performance optimization. Therefore, the peak power consumption of on-chip L2 caches in a CMP often needs to be constrained below a budget that is determined at runtime.

An effective way of controlling the power of L2 caches is to dynamically switch selected cache units (*e.g.*, ways, banks, or lines) between high- and low-power modes [7, 17]. For example, recent work [3] has proposed power-efficient management schemes for the non-uniform cache architecture (NUCA) designed to reduce wire delays. However, as a tradeoff, putting cache units into low-power modes may cause undesired long cache access latencies for the applications running on the CMP. As cache access latencies contribute significantly to the number of CPU stall cycles and thus are directly related to application performance such as IPC (instructions per cycle), how to select cache units for power capping is challenging. Simplistic solutions may allocate too few cache units to some applications, resulting in undesired thread starving and thrashing. Therefore, it is important to provide desired latency guarantees for the application threads running on the cores in a CMP, even when the active L2 cache size is being dynamically changed to enforce a runtime power budget.

There can be different ways to provide cache access latency guarantees in a power-constrained CMP. First, a straightforward way is to provide absolute latency guarantees for some applications based on their performance needs (*e.g.*, IPC requirements). However, since the active L2 cache size can be significantly reduced at runtime to enforce a low power budget in scenarios such as thermal emergencies, the available active cache size can become too small to guarantee the absolute latencies for any applications. Therefore, it may not be feasible to provide absolute latency guarantees in power-constrained CMPs. The second and a more reasonable way is to provide fairness guarantees such that the cache access latencies of the applications are impacted more uniformly by cache resizing. As a result, execution time fairness, *i.e.*, how uniform the execution times of co-scheduled threads are changed, can be better achieved. While existing work addresses fairness in terms of cache miss rate [14], we choose to control the average access latencies because the latency of each single access can vary significantly in NUCA caches due to wire delays. As a result, miss rate may not accurately indicate the impact on application performance.

Fair latency guarantees can also be extended to provide differentiated cache access latencies, since the applications co-scheduled on different cores in a CMP may have different performance needs or priorities. Differentiated latency guarantees can allow higher priority applications to have shorter

latencies and so less performance degradation under the impacts of cache resizing. Differentiated cache sharing is important because it can help the OS to enforce the desired thread priorities. The OS normally enforces thread priorities by assigning more time slices to higher priority threads, with the assumption that more time slices can lead to higher rates of progress among all the co-scheduled threads. However, as cache implementation is traditionally thread-blind and priority-blind, a novel scheme is needed to help higher priority threads to achieve shorter cache latencies and so higher rates of progress at the architectural level. Differentiated latency guarantees can also be useful in accelerating critical threads to reduce load imbalance. Existing work on thread criticality relies mainly on per-core dynamic voltage and frequency scaling (DVFS) and task stealing [4, 6]. However, some applications are not suitable for task stealing while some memory-intensive applications cannot be accelerated by per-core DVFS alone. In addition, per-core DVFS is not supported by most existing CMPs and can be expensive for future CMPs [23]. In this paper, we demonstrate that differentiated latency guarantees can provide another way to achieve desired rates of progress for application threads. As cache latencies may have small impacts on the performance of some applications, differentiated latency control should be combined with per-core DVFS and task stealing to accelerate threads based on application characteristics.

In this paper, we propose a novel L2 cache management strategy that can enforce the desired power budget for on-chip L2 caches in a power-constrained CMP by cache resizing. In order to achieve fair or differentiated cache sharing under the impacts of power capping, our strategy dynamically partitions the available active cache size among the threads on the cores for fair or differentiated cache access latencies. A key challenge in implementing fair or differentiated cache sharing in a power-constrained CMP is that the cache partition to achieve the desired degree of fairness or differentiation depends on two uncertain factors: workloads that are unknown a *priori* and the total cache size that is dynamically varying. A main contribution of this paper is the introduction of a coordinated feedback control architecture for adapting cache partitioning and resizing such that the desired latency fairness or differentiation among threads on different cores can be achieved, while the cache power consumption can be controlled. Specifically, this paper makes the following major contributions:

- We propose a novel L2 cache management strategy that provides fair or differentiated cache sharing for threads running on a CMP whose power consumption must be constrained. While prior research in this field focuses primarily on reducing the power consumption of L2 caches, to our best knowledge, this paper presents the first study to limit the peak power consumption of L2 caches.

- We design a two-tier coordinated feedback control architecture to simultaneously limit the peak cache power consumption and achieve the desired latency differentiation/fairness among different threads by dynamically conducting cache resizing and partitioning.

- We use advanced *feedback control theory* as a theoretical foundation to design and coordinate the two control loops for theoretically guaranteed control accuracy and system stability.

The rest of this paper is organized as follows. Section 2 introduces the proposed two-tier control architecture. Section 3 presents the modeling, design, and analysis of the latency fairness controller. Section 4 discusses the cache power controller. Section 5 provides the implementation details of our control solution. Section 6 presents our experimental results. Section 7 reviews the related work. Finally, Section 8 summarizes the paper.

2 Two-tier Cache Control Architecture

In this section, we provide a high-level description of our cache control system architecture, which features a two-tier controller design.

As shown in Figure 1, the primary control loop, *i.e.*, the L2 cache power control loop, limits the peak L2 cache power consumption under the desired budget (*e.g.*, 90% of the peak power consumption) by manipulating the total number of active L2 cache banks. Note that we use cache banks in NUCA caches as our actuation unit, but our controller can work with finer actuation granularities, such as cache lines or blocks, with only minor changes and slightly more hardware overhead. The key components in the cache power control loop include a *power controller*, a *power monitor*, and a *cache resizing* modulator. The control loop is invoked periodically. In each control period of the cache power control loop, the controller receives the total power consumption of the L2 caches from the power monitor. Note that although the power consumption of L2 caches cannot be directly measured in today's CMPs, it can already be precisely estimated with on-chip programmable event counters for control purpose [12]. The cache power is the *controlled variable* of this loop. Based on the difference between the measured power value and the desired power budget, the controller then computes the number of cache banks in the chip that should stay active in the next control period, which is the *manipulated variable*. The controller then sends the number to the cache resizing modulator. The modulator changes the power modes of selected cache banks accordingly. There is only one cache power control loop in a CMP.

The secondary control loop, *i.e.*, the latency fairness/differentiation control loop, achieves the desired latency fairness or differentiation for the threads on the cores by conducting cache partitioning at runtime on a smaller timescale. As shown in Figure 1, in a CMP with N CPU cores, we have $N - 1$ latency fairness/differentiation control loops. There are two ways to implement the latency control loops. The first way is that we select a reference thread on a core and then control the latencies of other threads relative to the latency of this reference thread. The second way is that

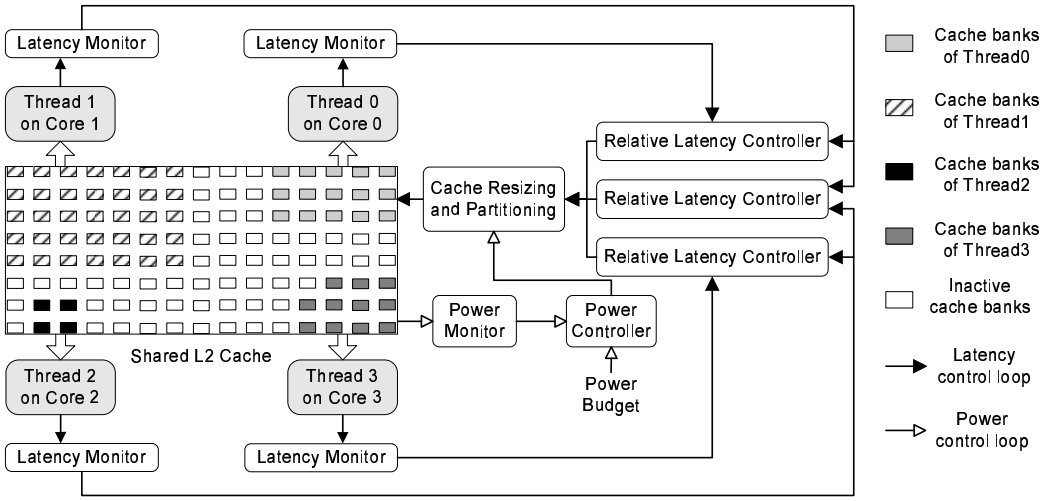


Figure 1. Two-tier L2 cache control architecture for power capping and fair/differentiated sharing.

we can have a control loop to control the latency ratio between the two active threads on every two adjacent cores. Since the controlled variable is the *relative* latencies among threads, the two implementations accomplish the same control functions. We plot the second way in Figure 1 for easier illustration. Specifically, to achieve fairness, we try to achieve approximately the same cache access latency for the active threads running on all the cores, despite the dynamically varying total cache size. Likewise, to achieve differentiation, we maintain the cache latency of a higher-priority thread shorter than that of a lower-priority thread. Note that previous fair cache sharing solutions [14, 34] try to manage the performance impacts of co-scheduling relative to the case when the applications are running alone on a CMP. As a result, they need to precisely know which applications are currently running on the CMP and then use corresponding off-line profiled performance measurements as references. A key advantage of our solution is that we do not assume such *a prior* knowledge and is thus more realistic. To handle multi-threaded applications, we can put cores running the same applications into groups and then conduct dynamic L2 cache partitioning among the groups.

The key components in a latency fairness/differentiation control loop for two cores include a *latency fairness/differentiation controller*, two *latency monitors* for the threads on the two cores, and a shared *cache partitioning* modulator. The control loop is also invoked periodically and its period is selected such that multiple cache access requests can be received during a control period and the actuation overhead is small compared with the period. The following steps are invoked at the end of every control period: 1) The latency monitors of the two cores measure the average absolute latencies of the two threads in the last control period and send the values to the controller through the feedback lane; 2) The controller calculates the relative latency of the two threads, which is the *controlled variable* of the control loop. The difference between the actual relative latency of the two threads and the desired value is the control error; 3) The controller then relies on control theory to compute the ratio between the numbers of cache banks to be allocated to the two threads, which is the *manipulated variable* of the control loop. The absolute number of cache banks can be calculated based on the total number of active banks determined by the power control loop; 4) The cache partitioning modulator then allocates the desired number of cache banks to each thread. Note that we assume a cache bank is not shared by multiple threads in this prototype design, but this assumption can be easily relaxed in a real implementation by using finer actuation granularities with more hardware overhead.

Clearly, the two control loops need to be coordinated based on advanced control theory, because every invocation of the cache power control loop may change the total number of active cache banks of the CMP, and thus affect the stability of the latency control loops. Therefore, in order to achieve stability for the two-tier control architecture, we adopt the method recently proposed in [31] to configure the period of the power control loop to be longer than the settling time of the latency fairness/differentiation control loops. This guarantees that the latency loop can always enter its steady state within one control period of the power control loop, so that the two control loops are decoupled and can be designed independently. As long as the two control loops are stable individually, the coordinated control system is stable.

Since the core of each control loop is its controller, we introduce the design and analysis of the two controllers in the next two sections, respectively. The details of cache resizing and partitioning in NUCA caches and the implementation details of other components in the control loops are discussed in Section 5.

3 Latency Fairness/Differentiation Controller

In this section, we present the modeling, design and analysis of the cache latency controller.

Given a CMP with N cores, as an example, we present the design process of the cache latency fairness/differentiation controller between Core i and Core j. The controller design between other pairs of cores is the same. We first introduce the following notation. $l_i(k)$ is the absolute access latency of the thread running on Core i in the k^{th} control pe-

riod. $l(k)$ is the relative latency ratio between the two threads on Core i and Core j. Specifically $l(k) = l_i(k)/l_j(k)$. L is the reference latency ratio between the two threads on Core i and Core j for the desired fairness (if $L = 1$). To achieve differentiation, the ratio L can be set (*e.g.*, by the OS) proportionally to the thread priorities or progress rates. It is important to note that L is not necessarily a constant and can be changed at runtime based on thread progress rates. For example, the thread criticality predictor in [4] can be used to predict how critical each of the co-scheduled threads is, such that proper reference ratios can be assigned to accelerate critical threads at different times. The focus of this paper is on providing a control scheme that can achieve the desired differentiation specified by the OS. We plan to investigate advanced algorithms for the OS to dynamically determine the reference ratio based on the desired thread priorities and/or progress rates in our future work. $e(k)$ is the difference between the reference ratio and the actual latency ratio. Specifically $e(k) = L - l(k)$. $c_i(k)$ is the number of active cache banks allocated to the thread on Core i in the k^{th} control period. $c(k)$ is the ratio between the numbers of cache banks partitioned to the two threads on Core j and Core i, specifically $c(k) = c_j(k)/c_i(k)$.

In the k^{th} control period, given the current access latency ratio $l(k)$, the control goal is to dynamically choose cache partition ratio $c(k)$ such that $l(k+1)$ can converge to the set point L after a finite number of control periods.

3.1 System Modeling

In order to have an effective controller design, it is important to model the dynamics of the controlled system, namely the relationship between the controlled variable (*i.e.*, $l(k)$) and the manipulated variable (*i.e.*, $c(k)$). However, a well-established physical equation is unavailable to capture the relationship between the latency ratio of two threads and the ratio of cache banks partitioned to them, due to the complexity of cache systems in today's computers. Therefore, we use a standard approach called *system identification* [8] to this problem. Based on control theory, we use the following standard difference equation to model the controlled system:

$$l(k) = \sum_{j=1}^{n_1} a_j l(k-j) + \sum_{j=1}^{n_2} b_j c(k-j) \quad (1)$$

where n_1 and n_2 are the orders of the system output (*i.e.*, $l(k)$) and system input (*i.e.*, $c(k)$), respectively. a_j and b_j are system parameters whose values need to be determined by system identification.

The model can be generally explained from the systems perspective. The current latency ratio between two threads on two cores, $l(k)$, is mainly determined by three factors: 1) the ratio $l(k-j)$ in the previous n_1 control periods, 2) the cache partitioning actions $c(k-j)$ happened in the previous n_2 control periods, and 3) the cache latency properties of the workloads, which are captured by the system parameters a_j and b_j through system identification. The system parameters may change for two reasons: 1) workloads may have phase changes at runtime, resulting in different latency properties and 2) the system may run different workloads. The variations of the system parameters must be theoretically analyzed to guarantee system stability.

For system identification, we need to first determine the right orders for the system, *i.e.*, the values of n_1 and n_2 in the difference equation (1). The order values are normally a compromise between model simplicity and modeling accuracy. In this paper, we test different typical system orders. For each combination of n_1 and n_2, we use white noise to generate a series of system inputs in a random fashion to stimulate the system and then measure the system output in each period. Based on the collected data, we use the *Least Squares Method (LSM)* to iteratively estimate the values of parameters a_i and b_i. Based on the method introduced in [22], we select four typical applications (*mcf*, *gap*, *ammp*, and *crafty*) in the SPEC CPU2000 benchmark suite as our workloads. *mcf* represents a workload with a data set larger than the available cache size. *gap* has a large number of compulsory misses. *ammp* continuously benefits as the cache size is being increased. *crafty* has a small working set but needs to frequently request data from L2 caches. We combine the four applications in 10 groups to conduct the system identification experiments. Based on our results, we choose to have the orders of $n_1 = 1$ and $n_2 = 1$ because this combination has a reasonably small error while keeping the model orders low. Therefore, the *nominal* system model resulted from our system identification is:

$$l(k) = a_1 l(k-1) + b_1 c(k-1) \quad (2)$$

where $a_1 = 0.595$ and $b_1 = 0.954$ are system parameters resulted from our experiments with the 10 groups of applications. To verify the accuracy of the model, we change the seed of the white noise signal to generate a different sequence of system inputs and then rerun the experiments to validate the results of system identification by comparing the actual system outputs and the estimated outputs based on the nominal model (2). Our results show that differences are sufficiently small.

Note that the real model of the system may be different from the nominal model (2) at runtime due to workload differences or parameter variations. To quantitatively analyze the impact of workload variations on the controller performance, in an extended version of this paper [29], we model the variations caused by different workloads and prove that the system controlled by the controller designed based on the nominal model can remain stable as long as the variation of a_1 is within the range of $[-0.69a_1, 1.18a_1]$.

3.2 Controller Design and Analysis

The goal of controller design is to achieve system stability, zero steady-state error, and short settling time. Following standard control theory [8], we design a *Proportional-Integral (PI)* controller to achieve the desired control performance. An important advantage of PI control is that it can provide robust control performance despite considerable modeling errors. In addition, PI control has a small computational overhead and is thus suitable to be implemented in

the cache system. A more sophisticated PID (Proportional-Integral-Derivative) controller is not used because it is determined by the system model (2) that a derivative (D) term is not needed for the desired control performance. Having a derivative term unnecessarily might amplify the noise in the cache latency ratio. The designed PI controller in the Z-domain is:

$$F(z) = \frac{K_p(z - K_i)}{z - 1} \tag{3}$$

where $K_p = 0.49$ and $K_i = 0.37$ are control parameters that are analytically chosen using the standard Root Locus design method [8]. The corresponding closed-loop poles are $0.5638 \pm 0.3247i$. Since both the poles are inside the unit circle in the complex plane, the system is guaranteed to be stable when the nominal system model (2) is accurate. Control performance such as system stability can be quantitatively analyzed when the system model varies due to workload variations, as discussed in Section 3.1. We have proven that the closed-loop system can precisely converge to the desired set point with a zero steady state error. The worst-case settling time of the closed-loop system is theoretically derived to be 7 control periods. The detailed proofs are skipped due to space limitations. The time-domain form of the designed PI controller is:

$$c(k) = c(k-1) + K_p e(k) - K_p K_i e(k-1) \tag{4}$$

Therefore, the computational overhead of the designed controller is just several multiplications and additions.

4 Cache Power Controller

In this section, we present the modeling, design, and analysis of the cache power controller.

We first introduce some notation. P is the desired power budget for the L2 caches in the CMP. $p(k)$ is the actual power consumption of the L2 caches in the k^{th} control period. $r(k)$ is the difference between the power budget and the actual power, specifically $r(k) = P - p(k)$. $s(k)$ is the total number of active cache banks in the k^{th} control period. $\Delta s(k) = s(k) - s(k-1)$ is the number difference of total active cache banks between the k^{th} and $k - 1^{th}$ control periods.

4.1 System Modeling

According to the investigation in previous work [26], cache leakage power has a linear relation with active cache size. The total power consumption of L2 caches is the sum of the leakage power and the dynamic power, with the leakage power as the predominant part. For example, recent research suggests using only leakage power to model the effect of cache resizing [16]. Another study also reports that dynamic cache power part contributes only a small portion to the total cache power consumption in a 4M uni-processor system [10]. Based on these observations, we analytically model the L2 cache power consumption as:

$$p(k) = c * s(k) + d \tag{5}$$

where c is a model parameter determined by the workload and d represents the dynamic power part. Similar to the system identification experiments in Section 3.1, we select four typical benchmarks (*mcf*, *gap*, *ammp*, and *crafty*) as our workloads to perform the experiments. In particular, we generate a series of system inputs (*i.e.*, $s(k)$) to stimulate the system and then measure the system output (*i.e.*, $p(k)$) in each control period. Based on the collected data, we use the *Least Squares Method (LSM)* to iteratively estimate the values of parameters c and d. Our results show that $c = 0.24$ on average for all the selected workloads. The value of d varies within a small range from 0.02 (with application *ammp*) to 0.58 (with application *mcf*). The results confirm the observation in [26, 10, 18] that leakage power dominates the total cache power consumption. d can be approximated as a constant because it is small compared to the leakage power part. The dynamic system model as a difference equation is:

$$p(k) = p(k-1) + c\Delta s(k) \tag{6}$$

Note that the real power model of the L2 caches may be different from the nominal model (6) at runtime due to several factors. For example, the leakage power of a cache bank can be sensitive to the system temperature. In the next subsection, we show that the closed-loop system controlled by a controller designed based on the nominal model (6) can remain stable as long as c varies within a certain range.

4.2 Controller Design and Analysis

The goal of the controller design is to achieve system stability, zero steady-state error, and short settling time when the nominal system model (6) is accurate. Similar to the design of the relative latency controller, we design a Proportional-Integral (PI) controller to achieve the desired control performance because of the robustness and negligible overhead of PI control. The time-domain form of the designed PI controller is

$$s(k) = s(k-1) + K_1 r(k) - K_1 K_2 r(k-1) \tag{7}$$

where $K_1 = 3$ and $K_2 = 0.73$ are control parameters that are analytically determined using the standard Root Locus design method. The corresponding closed-loop poles are $0.7349 \pm 0.2033i$. Since both the poles are inside the unit circle, the system is stable. We have also proven that the closed-loop system can precisely converge to the desired set point with a zero steady state error. The worst-case settling time of the system is 12 control periods.

Although the controlled system is guaranteed to be stable when the system model (6) is accurate, stability has to be reevaluated when the model parameter c changes. Following the three stability proof steps presented in [29], we have proven that the system can remain stable and achieve zero steady-state error when $c < -0.76$ or $c > 0$.

5 Implementation

In this section, we introduce our simulation environment and the implementation details of the control loops.

5.1 Simulation Environment

We extend the SimpleScalar simulator with *static*-NUCA cache configuration [19] as our simulation environment. In our simulations, we configure the CMP floor plan to have 4 Alpha 21264-like cores (65nm) with a frequency of 5GHz. We calculate the leakage power using HotLeakage [33] and calculate dynamic power using Wattch (built on CACTI) [5]. In all our experiments, we use the SPEC CPU2000 benchmark suite (V1.0) to test our two-tier cache control solution. The working temperature of the L2 cache is set to 80°C to simulate a typical real CMP setting based on the study in [9]. The L2 cache is configured to be 16MB (8-way set associative). The main memory latency is 300 cycles. The L2 cache replacement algorithm is LRU (Least Recently Used). The L1 i-cache and d-cache are both 32KB (2-way, 64 byte block size) with a 3-cycle hit latency. The detailed simulation configuration parameters are provided in the extended version of this paper [29].

5.2 Implementation of Control Loops

We now introduce the implementation details of each component in the two control loops.

Cache Access Latency Monitor. We add two counters for each core: one counter counts the number of stall cycles induced by cache accesses while the other counts the total number of cache accesses. We use the average access latencies in one control period to calculate the relative latency between the threads on two cores.

Cache Power Monitor. We use HotLeakage to calculate the leakage power consumption of the L2 caches and Wattch to calculate the dynamic power. We then use the sum of the two parts as our cache power reading. Note that the power consumption of L2 caches can be precisely estimated based on measurements feasible on physical chips [12].

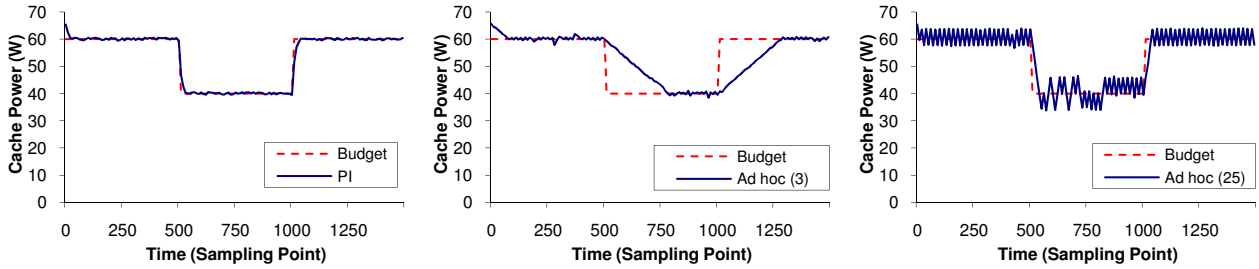
Controllers. As introduced in Sections 3 and 4, both the latency controllers and cache power controller are PI controllers. Each controller is invoked once in one of its control period to receive the measured value of its controlled variable (*i.e.*, relative latency or cache power) from the monitor, and then conduct its control algorithm to compute the corresponding control output. As shown in the controller functions (4) and (7), each invocation of the controller only executes a couple of multiplications and additions. Therefore, the computational overhead of the controllers is small enough to be implemented in a real cache system. The control period of the latency control loop is set to 1M CPU cycles as a compromise between the system response speed and actuation overhead. A longer control period may fail to promptly react to some system dynamics while a shorter period may lead to increased actuation overhead. Based on our stability analysis, the control period of the cache power control loop is set to 10 times that of the latency control loop, *i.e.*, 10M cycles, in order to decouple the two controllers for global system stability.

In a real system, there can be two ways to implement our controllers. First, as discussed in Section 3, the computational overhead of the two designed controllers is negligible. As a result, the controllers can be implemented on chip (as in Intel ItaniumII) to allow cache managment on a fine timescale. Second, our controllers can also be implemented in service processor firmware (as in IBM POWER7) and so their power and computational overheads will not directly affect the main CMP. This implementation is more flexible since the firmware is programmable but the control periods cannot be too small due to the communication delay between the service processor and the main CMP.

Cache Resizing and Partitioning Modulator. In order to implement cache resizing and partitioning for NUCA caches in a CMP with N cores, we add $\lg(N+1)$ flag bits for each cache bank, which have only negligible gate overhead to be implemented in real CMP chips. If a cache bank is allocated to Core i, we set the flag to the value i. A non-zero value indicates that the bank should be active. If the cache bank is not allocated to any core, we switch the cache bank to a low-power mode (*i.e.*, inactive). We use the Gated-Vdd technology [21] to implement the power mode switch of a cache bank. We assume that a cache bank is not shared by multiple threads in this prototype design, but a finer actuation granularity (*e.g.*, cache lines or blocks) is also available [13] and can be used in our control solution for even better control accuracy with a slightly higher hardware overhead.

Since the NUCA caches have non-uniform cache access latency related to the wire length, we have a cache resizing policy to ensure that the active cache banks of each core are always the ones that are closest to the cache I/O port of the core. We adopt this policy for three reasons. First, in NUCA caches, a cache bank that is farther away from the core normally has a longer access latency due to the longer wire. Second, a longer routing distance from the core may also lead to more routing hops, an increased probability of more contention, and thus reduced on-chip network routing capability. Third, if the dynamic data migration policy is enforced in the NUCA caches, cache banks that are far away from cores may get much less frequently accessed [2]. Therefore, switching those cache banks to the low-power mode will lead to considerable power savings with only slight impacts on cache performance.

After receiving the desired total active cache size from the power controller and the desired cache size ratios among threads on different cores from the latency controllers, the resizing and partitioning modulator calculates the number of active cache banks to be allocated to each thread. The modulator then uses the cache resizing policy to enforce the cache allocation. To implement the resizing policy, we maintain a table for each thread that records the cache banks allocated to the thread. In addition, in order to ensure that the active cache banks of a thread on a core are the ones that have the shortest distances to the core, the active cache area around the core needs to expand or shrink in both dimensions of the cache bank array. Therefore, we keep track of the cache row and column that are farthest from the core. Each core will expand or shrink in the farthest row and column alternatively to enforce the desired cache bank allocation. After the modulator allocates the active banks to the thread on



(a) PI cache power controller (b) Ad Hoc with 3 banks (b) Ad Hoc with 25 banks

Figure 2. Typical runs of the PI cache power controller and a baseline under a power budget reduction.

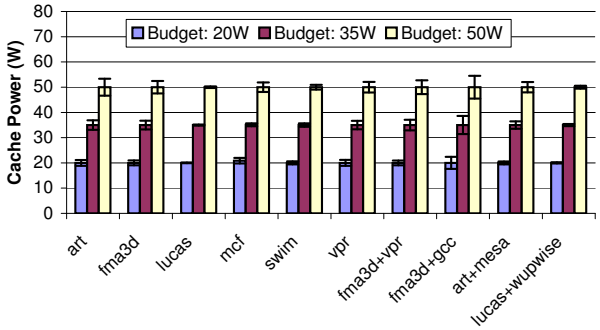


Figure 3. Control accuracy under different workloads and budgets.

each core, all the unallocated banks are switched to the low-power mode for power management. All the dirty cache lines in those banks are written back to the memory during the mode transition. Note that we assume static-NUCA instead of dynamic-NUCA (*i.e.*, D-NUCA) in this paper. The address mapping is modified according to the available cache size allocated to each core. The cache coherence is maintained by a snoopy bus protocol. We plan to evaluate our control solution with D-NUCA L2 cache in our future work.

6 Experimental Results

In this section, we present the results of our experiments conducted using the SimpleScalar simulator. We first examine the cache power controller to compare it with a heuristic-based baseline. We then test the two-tier cache control solution for fair and differentiated cache sharing in power-constrained CMPs. We randomly select a SPEC CPU2000 benchmark, *crafty*, as the default application in our experiments unless otherwise noted.

6.1 Cache Power Control

In this section, we examine the cache power controller without enabling the latency controllers. All the cache banks are initially active in this set of experiments.

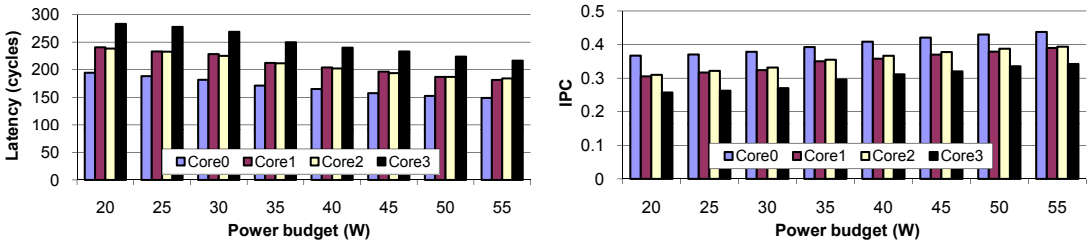
We first test the power controller in a common scenario where the power budget of the L2 caches needs to be reduced from 60 W to 40 W at the 500^{th} sampling point due to various reasons (e.g., thermal emergency). The power budget is then raised back to 60 W at the 1000^{th} sampling point after the emergency is resolved. This scenario is interesting because the power consumption of a CMP often needs to be capped and reduced at runtime [11, 30]. Reducing the power of L2 caches is an important way to maintain load balancing between CPU cores and L2 caches for optimized processor performance under the reduced budget. Figure 2(a) shows that the proposed PI power controller effectively controls the cache power to the desired budget by adjusting the active cache size. This experiment demonstrates that adjusting active cache size (and thus leakage power) is a promising way to conduct cache power management because leakage power contributes significantly to overall power consumption in large cache systems [18].

One may easily think that simplistic control solutions may also work well to control the cache power or achieve the desired fair or differentiated cache sharing. For example, a typical ad hoc power control solution (denoted as Ad hoc) would be to turn off M cache banks if the cache power violates its budget. However, without a theoretical foundation, it is commonly difficult to find a good value of M that would achieve the desired control performance in different cases. As shown in Figure 2(b), if M is too small (*e.g.*, 3 banks), Ad hoc may take an unnecessarily long settling time for power to converge back to the budget, resulting in overheating and even undesired processor shutdown. On the other hand, as shown in Figure 2(b), if the step size M is too large (*e.g.*, 25), Ad hoc may have large oscillations and overshoots. Therefore, it is undesirable to use Ad hoc in practice, which is consistent with the observations in [28, 30]. A fundamental benefit of the control-theoretic approach is that it provides standard ways to choose the right control parameters and gives us confidence for system stability and control accuracy. In addition, naive control solutions may also cause different control loops to conflict with each other.

The cache power controller is designed based on a system model whose parameters are the results of system identification. Therefore, in order to test the robustness of the power controller when it is used in a system that is running a different workload, we conduct a set of experiments with different workload combinations under different power budgets: 20W, 35W, and 50W. Figure 3 shows that the average power of the L2 caches can be precisely controlled to the desired budget (with the maximum standard deviation smaller than 2% of the budget). The experimental results demonstrate that the power controller can precisely control the power consumption of the L2 caches for different workloads and power budgets with only small deviations.

6.2 Power-Performance Tradeoffs

In the previous sections, we have demonstrated that the cache power controller achieves the desired control function



(a) Cache access latencies (b) IPC values

Figure 4. Cache access latencies and IPC values of the four threads (running *crafty*) on the four cores of the CMP. When power budget varies from 20W to 55W, all the threads have reduced cache latencies and increased IPCs, while the desired latency differentiations and fairness are maintained.

individually. In this section, we enable all the controllers to demonstrate that our two-tier control solution can achieve desired power-performance tradeoffs. For each benchmark, we first run 2 billion instructions and then start to measure the cache access latency and IPC values for the next 100 control periods of the power control loop (*i.e.*, 1 billion cycles).

In the first experiment, we run application *crafty* on all the 4 cores. To investigate the impacts of power constraint on fair and differentiated cache sharing, we increase the power budget from 20W to 55W with an increment of 5W. The cache sharing priority of Cores 0, 1, and 3 is set to *high, medium*, and *low*, respectively, to test the differentiated cache latency guarantees, while the priority of Core 2 is also set to *medium* to test fair cache sharing in terms of cache latency. Figure 4(a) shows that all the four threads on the four cores have reduced cache latencies when the power budget increases from 20W to 55W, due to increased number of total active cache banks. As a result, Figure 4(b) shows that all the threads have increased IPC values. In the meantime, the desired cache latency priorities have been strictly enforced, allowing the OS to achieve the desired rates of progress among the co-scheduled applications, despite that all the threads have improved performance with the increased power budget. In particular, by setting the same priority for Cores 1 and 2 for fairness, the latency differences between Core0 and Core1 are less than 5 cycles and the IPC differences are less than 0.01 under all the power budgets. This demonstrates that fair sharing, as a special case of differentiated sharing, can be guaranteed by our control solution. As discussed in Section 1, differentiated sharing can help the OS to enforce the desired thread priorities at the architectural level.

The benchmark used in this experiment, *crafty*, is a chess game playing application and is one of the SPEC integer benchmarks. *crafty* represents applications with a significant number of logical operations that are relatively simple but need to frequently request data from the caches. As a cache-sensitive application whose execution is dominated by cache accesses [20], *crafty* achieves an almost reverse linear relationship between cache latency and IPC, *i.e.*, the decrease rate of the cache latency is approximately proportional to the increase rate of the IPC. While this experiment demonstrates the effectiveness of our control solution with cache-intensive benchmarks, it is important to investigate other benchmarks with different cache-access properties.

In the second set of experiments, we conduct the same experiment with different benchmarks for three power budgets: 20W, 35W, and 50W. Our workloads include memory-intensive benchmarks such as *apsi*, *art*, *bzip2*, and *swim*, as well as randomly selected benchmarks. We test two kinds of workload combinations: 1) four copies of a single benchmark running on the 4 cores and, 2) one benchmark running on Cores 0 and 2 while the other benchmark running on Cores 1 and 3.

To save space, we only present the results of the two threads on Cores 0 and 1. Cores 0 is configured to have a higher cache sharing priority than Core 1. As shown in Figure 5(a), when the power budget increases from 20W to 50W, the cache latencies of the two threads on Cores 0 and 1 decrease for all the benchmark combinations and decrease in an approximately linear way for many benchmark combinations, such as *bzip2+art*, *twolf*, and *gap+galgel*. Figure 5(b) shows that the IPC values of the two cores increase as the power budget increases for all the benchmark combinations. In addition, the Core0 thread has a higher IPC value than the Core1 thread for most benchmark combinations, except *mgrid+mesa*. Since *mgrid* and *mesa* have significantly different cache-access properties, the Core1 thread (running *mesa*) has a higher IPC than the Core0 thread (running *mgrid*) though it has a longer latency. Note that in some mixed benchmark cases similar to *mgrid+mesa*, a higher IPC may still be achieved for the Core0 thread by allowing the OS to dynamically adjust the reference set point at runtime based on the measured thread progress or combining per-core DVFS or task stealing. The experiments demonstrate that our two-tier control solution can be used to achieve the desired power-performance tradeoffs and maintain performance differentiations at the same time.

It is important to note that the controllers may saturate for some benchmark combinations. For example, although the two-tier control solution can precisely achieve the desired latency priorities for most benchmark combinations, the latency controller saturates for some benchmarks, such as *mesa* when the power budget is 20W. Note that it is easier for the latency controller to saturate under small power budget, because when the power budget is small, the number of active cache banks is small, resulting a limited range for the latency controller to conduct dynamic cache partitioning. Consequently, it may become infeasible for the latency controller to achieve the desired priorities. Figure 5(a) shows that the Core0 thread only has a slightly shorter cache la-

tency than the Core1 thread for *mesa* due to saturation when the power budget is 20W. As a result, the IPC values of the two threads are close in this case. However, when the power budget increases to 35W and 50W, the latency controller has more active cache banks for partitioning and thus can achieve the desired latency priorities by allocating more cache to the high-priority thread (*i.e.*, the Core0 thread). Consequently, the Core0 thread can have a much shorter cache latency and thus a much higher IPC value than the Core1 thread.

We acknowledge that cache latencies may have small impacts on the performance of some applications (*e.g.*, CPU-intensive ones). The goal of our paper is to demonstrate that differentiated latency control can provide another way to achieve desired rates of progress for some co-scheduled application threads (*e.g.*, cache-intensive ones). In a real system, differentiated latency control should be combined with per-core DVFS and task stealing to accelerate threads based on application characteristics, which is our future work.

7 Related Work

Power has become an important design constraint for CMPs. Some recent work has studied power capping for CMPs. For example, Intel's Foxton technology [24] has successfully controlled the power and temperature of a processor by using chip-wide DVFS. Isci et al. [11] proposed both a closed-loop algorithm and a prediction based algorithm to control the power of a CMP to stay below a power budget. Meng et al. [16] presented heuristic-based algorithms to use both per-core DVFS and cache resizing as actuators to control power. Teodorescu et al. [27] developed an optimization algorithm based on linear programming to provide power management for a CMP based on both DVFS and thread mapping. Wang et al. [30] developed a control-theoretic power controller for improved system performance. While the related work mainly focuses on adapting the dynamic power of the CPU cores in a CMP, we use cache resizing to control the cache power and provide performance differentiations. Our scheme can be combined with those CPU core power control solutions for chip-level power management.

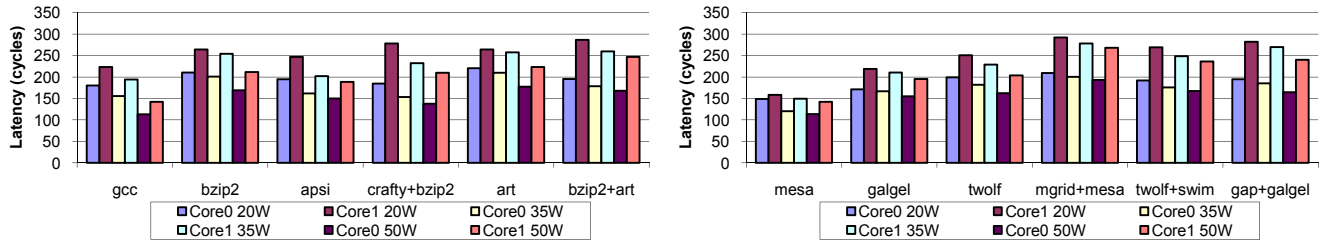
Adaptive cache partitioning for CMPs has recently received a lot of attention. This paper is different because we use cache partitioning to achieve performance differentiation in NUCA caches for threads with different performance needs and priorities in a CMP with power constraints. Some recent work has been proposed to dynamically adapt the cache size for power savings. For example, Albonesi et al. proposed to turn off cache ways for reduced dynamic power [1]. Bardine et al. explored the possibility of applying this technique to NUCA caches [3]. Kobayashi et al. presented a heuristic-based control algorithm based on locality metrics [15]. Our paper is different because 1) we provide differentiated performance guarantees in addition to power management, and 2) we rely on control theory as a theoretical foundation for theoretically guaranteed control accuracy and system stability.

Feedback control theory has been successfully applied to control temperature, power, and performance in computer architecture research. For example, Skadron et al. [25] used control theory to dynamically manage the temperature of microprocessors. Likewise, Wu et al. [32] managed power using dynamic voltage scaling by controlling the synchronizing queues in multi-clock-domain processors. In contrast to their work that relies on basic control theory to design a single control loop, we coordinate two control loops in a hierarchical way for guaranteed stability.

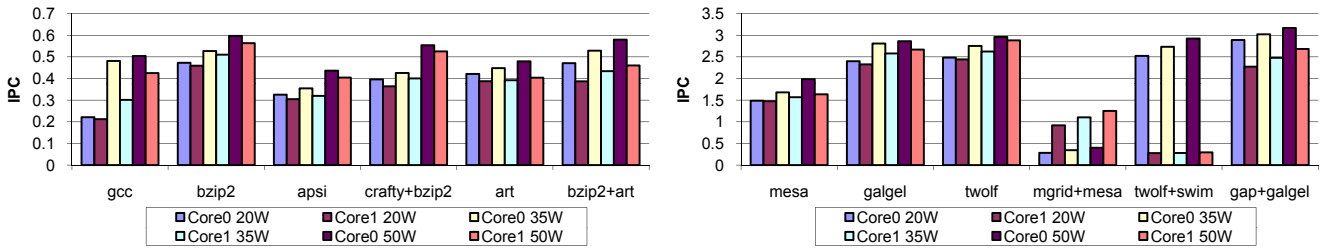
Our work is also related to thread criticality research. Bhattacharjee et al. [4] proposed a novel way to predict thread criticality. Cai et al. [6] used DVFS to slow down non-critical threads for power savings. In our work, we provide cache latency differentiations to accelerate critical threads. Our scheme can also be used to guarantee cache sharing fairness. While Kim et al. proposed partitioning policies with five fairness metrics defined based on cache miss rates [14], we address average latencies because the latency of each single access can vary significantly in the NUCA caches. As a result, miss rate may not accurately indicate the impact on application performance. Zhou et al. [34] also proposed fair cache sharing to have the same impacts on execution times. However, similar to [14], they need to precisely know which applications are currently running on the CMP such that they can use corresponding off-line profiled execution times as references.

8 Conclusions and Future Work

In order to enable chip-level power capping, the peak power consumption of on-chip L2 caches in a CMP often needs to be constrained by dynamically transitioning selected cache banks into low-power modes. While prior research in this field focuses primarily on reducing the power consumption of L2 caches, this paper aims at limiting the peak power consumption of L2 caches. To avoid undesired thread starving and thrashing caused by dynamic cache resizing for power capping, our strategy can provide fairness guarantees such that the cache access latencies of the application threads co-scheduled on the CMP are impacted more uniformly. Furthermore, our strategy is also extended to provide differentiated cache latency guarantees that can help the OS to enforce the desired thread priorities at the architectural level and achieve differentiated rates of thread progress for co-scheduled applications. Our solution features a two-tier control architecture rigorously designed based on advanced feedback control theory for guaranteed control accuracy and system stability. Extensive experimental results demonstrate that our solution can achieve the desired cache power capping, fair or differentiated cache sharing, and power-performance tradeoffs for many applications. Our results also demonstrate that differentiated latency guarantees can provide another effective way to achieve desired rates of progress for application threads. As cache latencies may have small impacts on the performance of some applications, differentiated latency control should be combined with per-core DVFS and task stealing to accelerate threads based on application characteristics, which is our future work.



(a) Cache access latencies. Reduced latencies and desired latency differentiations are achieved with increased power budget.



(b) IPC values. Increased IPC values and desired progress differentiations are achieved with increased power budget.

Figure 5. Cache access latencies and IPC values of the higher-priority thread on Core 0 and the lower-priority thread on Core 1 under different benchmarks, when the power budget increases from 20W to 50W.

Acknowledgements

We thank Naveen Muralimanohar at HP Labs for providing the source code of SimpleScalar S-NUCA cache implementation and anonymous reviewers for their valuable comments. This work is funded in part by NSF under grants CNS-0720663, CNS-0845390, CNS-0915959, and CCF-1017336, and by ONR under N00014-09-1-0750.

References

[1] D. H. Albonesi. Selective cache ways: on-demand cache resource allocation. In *MICRO*, 1999.

[2] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli, C. A. Prete, and P. Stenström. Leveraging data promotion for low power D-NUCA caches. In *DSD*, 2008.

[3] A. Bardine, P. Foglia, G. Gabrielli, C. A. Prete, and P. Stenström. Improving power efficiency of D-NUCA caches. *SIGARCH Comput. Archit. News*, 35(4), 2007.

[4] A. Bhattacharjee and M. Martonosi. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. *SIGARCH Comput. Archit. News*, 37(3), 2009.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA*, 2000.

[6] Q. Cai, J. González, R. Rakvic, G. Magklis, P. Chaparro, and A. González. Meeting points: using thread criticality to adapt multicore hardware to parallel regions. In *PACT*, 2008.

[7] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *ISCA*, 2002.

[8] G. F. Franklin, D. J. Powell, and M. Workman. *Digital Control of Dynamic Systems, 3rd edition*. Addition-Wesley, 1997.

[9] A. Greenhill. *Power Saving in the UltraSPARC T1 Processor*. Sun Microsystem Whitepaper, 2005.

[10] H. Homayoun and A. Veidenbaum. Reducing leakage power in peripheral circuits of L2 caches. In *ICCD*, 2007.

[11] 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 *MICRO*, 2006.

[12] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO*, 2003.

[13] S. Kaxiras, Z. Hu, G. J. Narlikar, and R. McLellan. Cache-line decay: A mechanism to reduce cache leakage power. In *PACS*, 2001.

[14] S. Kim, D. Chandra, and Y. Solihin. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *PACT*, 2004.

[15] H. Kobayashi, I. Kotera, and H. Takizawa. Locality analysis to control dynamically way-adaptable caches. *SIGARCH Comput. Archit. News*, 33(3), 2005.

[16] K. Meng, R. Joseph, R. P. Dick, and L. Shang. Multi-optimization power management for chip multiprocessors. In *PACT*, 2008.

[17] Y. Meng, T. Sherwood, and R. Kastner. Exploring the limits of leakage power reduction in caches. *ACM Trans. Archit. Code Optim.*, 2(3), 2005.

[18] Y. Meng, T. Sherwood, and R. Kastner. On the limits of leakage power reduction in caches. In *HPCA*, 2005.

[19] N. Muralimanohar and R. Balasubramonian. Interconnect design considerations for large NUCA caches. In *ISCA*, 2007.

[20] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *MICRO*, 2007.

[21] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-Vdd: a circuit technique to reduce leakage in deep-submicron cache memories. In *ISLPED*, 2000.

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

[23] K. R. Rangan, G.-Y. Wei, and D. Brooks. Thread motion: Fine-grained power management for multi-core systems. In *ISCA*, 2009.

[24] M. Rich, P. Christopher, C. Bostak, J. Ignowski, M. Millican, W. H. Parks, and S. Naffziger. Power and temperature control on a 90-nm itanium family processor. *IEEE Journal of Solid-State Circuits*, 41(1), 2006.

[25] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *HPCA*, 2002.

[26] K. Skadron, M. R. Stan, et al. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.*, 1(1), 2004.

[27] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *ISCA*, 2008.

[28] X. Wang and M. Chen. Cluster-level feedback power control for performance optimization. In *HPCA*, 2008.

[29] X. Wang, K. Ma, and Y. Wang. Achieving Fair or Differentiated Cache Sharing in Power-Constrained Chip Multiprocessors, Tech Report, EECS Department, University of Tennessee. http://pacs.ece.utk.edu/techreports/tech1018.pdf, 2010.

[30] Y. Wang, K. Ma, and X. Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. In *ISCA*, 2009.

[31] Y. Wang, X. Wang, M. Chen, and X. Zhu. Power-efficient response time guarantees for virtualized enterprise servers. In *RTSS*, 2008.

[32] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, and D. W. Clark. Formal control techniques for power-performance management. *IEEE Micro*, 25(5), 2005.

[33] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Tech Report, Univ. of Virginia, 2003.

[34] X. Zhou, W. Chen, and W. Zheng. Cache sharing management for performance fairness in chip multiprocessors. In *PACT*, 2009.