Computer Sciences Department

A Case for an Over-provisioned Multicore System: Energy Efficient Processing of Multithreaded Programs

Koushik Chakraborty
Philip M. Wells
Gurindar S. Sohi

Technical Report #1607

October 2007

UNIVERSITY OF WISCONSIN MADISON

A Case for an Over-provisioned Multicore System: Energy Efficient Processing of Multithreaded Programs

Koushik Chakraborty
University of Wisconsin-Madison
kchak@cs.wisc.edu

Philip M. Wells
University of Wisconsin-Madison
pwells@cs.wisc.edu

Gurindar S. Sohi
University of Wisconsin-Madison
sohi@cs.wisc.edu

ABSTRACT

Technology scaling has provided system designers with an exploding transistor budget, far more than what was available when the core principles behind many existing commodity microprocessors were envisioned. With this tremendous growth, however, comes a whole new set of engineering challenges involving power density, thermal efficiency, programmability and so on. In this paper, we study another important trend in high performance microprocessors: the reduction in the Simultaneously Active Fraction (SAF) — the fraction of the entire chip resources that can be active simultaneously, given a target power envelope. As the improvement in the energy efficiency of individual transistor devices is lagging behind the growth in their integration capacity, we find that the SAF is monotonically decreasing for each successive technology generation.

Given this increasing constraint on the SAF, we examine the utility of temporarily suspending computation on a core as a means for reducing the SAF, and hence, remain within the confines of cost-effective cooling and power delivery. We investigate a SAF aware over-provisioned multicore system (OPMS), where only a subset of the available cores are employed to perform active computation at any given time, by allowing the individual cores to transition between active and inactive state. Though several possible directions for utilizing such an over-provisioned system are possible, this paper focuses on energy efficient dynamic task redistribution. In particular, this paper examines the use of Computation Spreading—a recently proposed technique for runtime specialization of homogeneous multicores—in an OPMS. We show several benefits for such an OPMS design, including reductions in energy, runtime, and superior thermal characteristics. Overall, our technique improves the energy-delay product of the commercial workloads we examine by 5–20%.

1. INTRODUCTION

Several contrasting technology trends are likely to shape the design of future microprocessors and the low level software, such as operating systems and *Virtual Machine Monitor (VMM)*, executing on them. On one hand, we are experiencing a resource explosion due to the exponential increase in transistor density: ITRS predicts that high performance microprocessors are likely to contain more than 4 billion transistors within a couple of technology generations from today (by 32nm) [18]. On the other hand, several engineering challenges are imposing substantial constraints on viable microprocessor designs. For example, growing geo-political and economic concerns about energy usage are driving the demand for more power efficiency [18, 33], but the power density of hardware components continues to increase with every new process generation due to the inability to achieve sufficient reduction in supply voltage [20]. Since the relative increase in energy efficiency of individual transistors is lagging behind the growth of their aggregate volume, the power dissipation of a chip is already reaching the practical limits for cost-effective cooling. Going forward, we expect that the portion of on-chip hardware that can be active simultaneously — the *Simultaneously Active Fraction (SAF)* — will continue to decrease so as to remain within a reasonable power envelope. This gradual fall of SAF will have significant impact on several key design principles of future multicore systems. Low level system software, which often requires an intricate knowledge of the underlying architecture, and are increasingly responsible for power management [38, 43], will also have to cope with this downward trend.

Various circuit techniques are already in place to achieve energy efficiency by effectively decreasing the SAF. Dynamic voltage and frequency throttling, fine-grain clock gating within a processor core, and disabling portions of the cache storage are some such examples. But, given the downward trend in the SAF, as well as diminishing returns from previously effective schemes like voltage scaling going forward [9, 13], future systems are likely to strive for further SAF reduction at *multiple* levels in their design. An orthogonal approach to these current circuit techniques is to temporarily suspend the computation on an entire processing core. Consider, for example, a multicore system where only a subset of the available processor cores are performing computation simultaneously, while the rest are kept in a low-leakage sleep state. This design paradigm contrasts against the conventional systems where processing cores are typically in continuous use. The key question, then, is whether this novel paradigm of reduced utilization of individual processor cores—components traditionally designed for high utilization—can collectively comprise an efficient system and execute unmodified complex multithreaded applications.

In the light of these technology and design trends, this paper examines the energy efficiency of a multicore design, managed by a lightweight VMM, where only a subset of the available processing cores are utilized at any given time. In this design, which we refer as an *Over-provisioned Multicore System (OPMS)*, the number of available processor cores exceeds the maximum number of simultaneously active cores (at the nominal voltage-frequency) allowed by the target power envelope. We believe that such an OPMS will enable a variety of coarse grain architectural techniques to mitigate several design challenges and expose many interesting opportunities to the rapidly evolving virtualization technology. In this paper, we employ computation phase-driven dynamic task re-distribution in multithreaded commercial workloads using a lightweight VMM. We use a recently proposed technique by Chakraborty, et al., to identify appropriate phases in our commercial workloads [8].

This paper makes several contributions:

- First, we present a quantitative analysis of the constraints imposed by cost-effective cooling techniques on future microprocessor designs. Based on the expected performance characteristics of transistor devices, we estimate the *Simultaneously Active Fraction* for several future technology generations. We find that this SAF is monotonically decreasing, despite several key technology innovations to improve power efficiency. This gradual fall of SAF is likely to significantly alter the design considerations for high performance multicore systems and low-level system software.
- Second, we extend the concept of activity suspension, typically employed at a fine grain, beyond the confines of a single core to temporarily suspend computation on an entire core. We propose a novel paradigm of SAF aware multicore design, referred as *Over-provisioned Multicore System*, where processing cores transition between active and low-leakage sleep state. In this design, while the pool of core resources exceeds the maximum number of software contexts that can be active simultaneously, a lightweight VMM is employed to ensure that the overall chip remains within the target power envelope.
- Third, one particular application of such an over-provisioned multicore — energy-efficient processing of multithreaded workloads — is evaluated in detail. This technique selectively utilizes the available on-chip cores using *Computation Spreading*, a recently proposed technique to specialize homogeneous multicore by dynamically reassigning computation from multi-threaded server workloads [8]. Apart from the baseline system which fully utilizes the core resources, we also compare our proposal with a slightly modified version of Heat and Run designed for an OPMS [14]. We find that an OPMS employing CSP_O improves the energy-delay product for our workloads by 5–20%, while providing similar thermal benefits as Heat and Run.

The rest of the paper is organized as follows. In Section 2, we present our analysis of the SAF in future microprocessor designs. We introduce the concept of over-provisioned multicore system and argue how it can be used in a variety of ways to improve overall system efficiency in Section 3. Section 4 provides the implemention details. Section 5 describes our experimental methodology and we discuss our key results in Section 6. In Section 7, we discuss related work, and then conclude in Section 8.

2. TECHNOLOGY IMPACT ON SIMULTANEOUSLY ACTIVE FRACTION (SAF)

2.1 Simultaneously Active Fraction (SAF)

The limitations of cost effective cooling techniques, along with the growing energy dissipation due to aggressive technology scaling, have already imposed severe challenges to microprocessor designers. Several studies have shown how the increasing power density of future generation processors will easily cross over the capabilities of reasonable cooling solutions [6, 29]. Economic and engineering challenges to cool enterprise data centers are also driving the need for energy efficiency in a single-chip — to an even greater extent than what is required by the practical cooling limits for an individual multicore in isolation [33]. Given this technology trend and real-world challenges, we study the impact on the *Simultaneously Active Fraction (SAF)* — the fraction of the entire transistor budget that can be simultaneously active in a microprocessor designed for a particular power envelope.

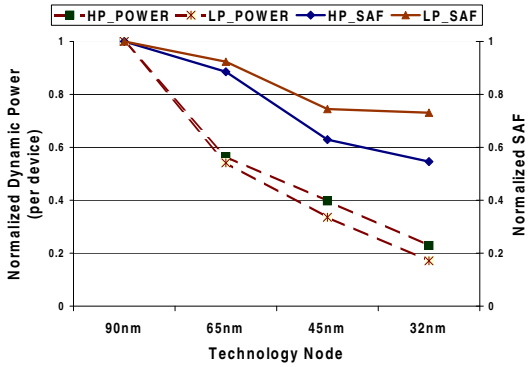

Figure 1: Dynamic power consumption trend per device and SAF due to dynamic power exclusively

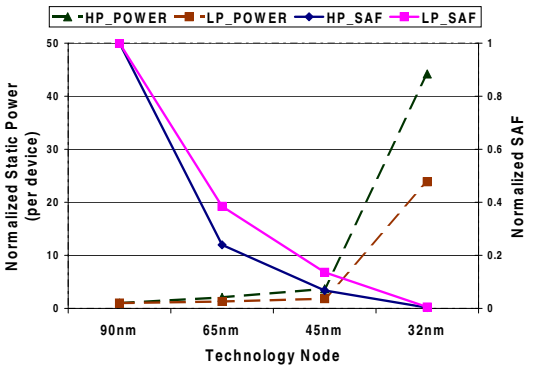

Figure 2: Static power consumption trend per device and SAF due to static power exclusively

2.2 SAF Trends: A Quantitative Analysis

Several technology assumptions are necessary for a quantitative analysis of the SAF trend. For the sake of simplicity, let us consider the performance characteristics of two typical N-MOSFET devices, one designed for high-performance (HP), and the other for low power (LP). HP devices use a low threshold voltage and a nominal supply voltage, while the LP devices use a higher threshold voltage and a scaled supply voltage. Together they encompass the range of devices, with diverse operating conditions, typically seen in high-performance microprocessors [19]. For an estimate of the SAF, we further assume that the entire chip is built with these devices, and that the chip size remains constant, which results in the doubling of aggregate devices every generation [18]. Based on the power consumption of a single device, determined using the MASTAR tool [21], we compute the SAF at each technology node as the fraction of the aggregate devices that account for a given target power envelope. The power envelope is assumed to remain constant across technology generations. The results are shown in Figures 1 and 2, and are normalized to the 90nm technology node in order to better understand the trend.

The trend in dynamic power, and the resulting SAF when considering dynamic power alone, is shown in Figure 1. We assume a 17% improvement in switching frequency in successive generations based on the ITRS guidelines for logic components [19]. HP_POWER and LP_POWER show the power consumption of a single HP and a LP device, respectively, while HP_SAF and LP_SAF show the SAF for HP and LP devices, respectively. We

notice that the improvement in dynamic power efficiency is less than the improvement in device integration capacity, both for the high performance and low power devices. Consequently, we notice a fall in SAF in each successive generation, where the relative drop in SAF is inversely proportional to the improvement in the power efficiency of individual devices. The fall of SAF is more pronounced in the HP devices than LP due to comparatively smaller scaling in the supply voltage. In summary, we expect that dynamic power will continue to limit ways to use aggregate resources, even when considered in isolation, due to the gradual reduction of SAF.

Static power due to leakage is an additive power component in microprocessors. We show the trend in static power per device in Figure 2, along with the SAF when considering only static power. We notice a dramatic reduction in SAF when considering static power, as unlike dynamic power, the static power of individual devices is *increasing* in successive technology generations. Although we do expect that leakage-induced static power will be mitigated by employing various circuit techniques such as high κ-dielectric [19], FINFET devices [3], etc., it is still likely to also pose substantial constraints on aggregate resource usage.

While the analysis above provides a relative SAF trend, the actual SAF of a real implementation depends on various other factors not considered in this analysis. The static and dynamic power consumption depends on a variety of device level parameters, which vary widely across the entire chip. On a chip-wide scale, the energy consumption is further complicated by several other implementation specific details such as circuit design style, whether hardware resource is provisioned for cache or cores, the percentage of transistors in the latency critical path, and the temperature and activity distribution. Despite these issues, and in the absence of dramatic alteration of device properties, we expect that this downward trend of the SAF will persist.

2.3 Techniques for Reducing SAF

From the discussions above, it is apparent that for any given chip design, the SAF has a reasonably tight upper bound as determined by the device characteristics and the packaging constraints. With the expanding transistor budget, reducing the SAF, without causing a significant reduction in performance, is a key design goal for an energy efficient implementation. Currently, different techniques are applied to caches and processor cores — two major components in a modern multicore system.

Caches.

By design, the second or third level (L2 or L3) caches have low activity as only a small fraction of the aggregate cache lines are accessed in any given cycle. In addition, large sections of the caches can be put into a sleep state, to reduce both leakage and dynamic power [34]. However, increasing cache resources instead of providing more cores—an effective ploy for reducing the SAF—is unlikely to provide a commensurate performance boost. Recent studies demonstrate the diminishing returns for large L2 caches for commercial workloads [5, 36], as well as parallel bio-informatics data mining workloads [22].

Cores.

Since the processor cores often consume the bulk of the power budget (for example, in [34]), several past proposals advocate circuit techniques for reducing the activity through fine-grain clock-gating. Clock gating temporarily suspends the switching activity in the transistors which are not part of any computation in a given cycle. Similar techniques can be applied to reduce leakage power as well (e.g., dynamic sleep transistors [40]). Consequently, while a large pool of transistors comprise the heavily active hardware components, over a smaller timescale, only a subset of them contribute to the SAF of the chip.

Fundamentally, this temporary suspension of activity is the key for exploiting a growing pool of resources, while honoring the constraints imposed by the shrinking SAF. In this paper, we take an orthogonal approach to these circuit-level techniques, and investigate the implications of temporarily suspending computation on an entire processor core. We propose to extend the concept of activity suspension beyond the confines of a single core using a design that we call an *Over-provisioned Multicore System (OPMS)*. However, as we extend the concept of activity suspension to an entire core, which is typically the lowest granularity of compute resources managed by system software or a VMM, this paradigm of multicore design can have substantial impact on the future system software or VMM. In this paper, we demonstrate how a lightweight VMM can exploit an OPMS design, without requiring any modification to server class multithreaded workloads and the operating system we study.

3. AN OVER-PROVISIONED MULTICORE SYSTEM

An *Over-provisioned Multicore System (OPMS)* is simply a SAF-aware design alternative, where the number of available processing cores exceeds the number of simultaneously active cores allowed by the power or thermal limits of the chip. At any given time, certain cores are actively executing code, while other cores are held in a low-power sleep state. Consequently, the aggregate number of physical cores on the chip can exceed the allowable number of simultaneously active cores operating at the nominal voltage-frequency. Execution of a multithreaded workload is spread throughout the cores, such that at various times, different cores are active. Unlike a single-core design, where turning off parts of the core was the main option, especially components that are over-provisioned to efficiently support a diverse application domain (like the issue width, the physical register file, or instruction window entries), this coarse grain architectural technique now becomes possible in a multicore system. Such a design contrasts current implementations, where core resources are provisioned for continuous and simultaneous use.

In an OPMS, an individual core transitions between active and inactive states. In an active state, a core performs computation as normal. In an inactive state, a core does not perform computation, but continues to be useful, for example, by retaining predictive states, allowing its critical hotspots to cool off, or simply existing as a spare. Consequently, an OPMS has a time-varying pool of physical computation resources, which may be homogeneous or heterogeneous. The first challenge is to determine how to use an OPMS without placing onerous demands on the software. We propose to do this via *dynamic task reassignment*, where a lightweight VMM is employed to map *computation fragments* (i.e., portions of a software thread) to the physical processing cores (see Section 4.1).

Given the framework of an OPMS with dynamic task reassignment, we can then consider how an OPMS can be used to improve computation and energy efficiency, thermal management, and lifetime reliability. In this paper, we focus on using an OPMS to improve energy efficiency, though we also explore the thermal management qualities of such task reassignment. We leave exploration of lifetime reliability for future work.

3.1 Using an OPMS for Energy Efficiency

Performing task redistribution in an OPMS can improve efficiency by utilizing the most appropriate core for any given *computation fragment* (a portion of a software thread). In an OPMS with heterogeneous processing resources, processors with different capabilities could be used to execute different program phases, generalizing the technique proposed by Kumar, et al. [23], where they detect the varying ILP characteristics of a single-threaded program and move the computation among several cores with various ILP extracting capabilities (and, of course, various power requirements). In a system with homogeneous processing cores, task redistribution can boost both compute *and* energy efficiency, by specializing the predictive components of each core (i.e., caches and branch predictors), such that a particular computation fragment executes on a core that retains most of its predictive state. While a conventional architecture attempts to efficiently use the predictive structures by coarse grain task assignment, the conflicting nature of computation fragments within a task (e.g. user and OS code) disrupts its intended goal [8]. In both cases, the utilization of an individual core is reduced, but the efficiency of the system as a whole increases as a fewer number of aggregate transistor resources participate in carrying out the overall computation.

In this paper, we consider a homogeneous OPMS running multi-threaded, server class workloads. The first step then is to develop a framework for distributing computation fragments from the given workload across the processing cores.

3.1.1 Computation Spreading

The idea of dynamic task redistribution using a lightweight VMM, for a traditional, fully-provisioned system, was recently proposed by Chakraborty, et al., as *Computation Spreading (CSP)* [8]. In their model, the OS schedules a collection of software threads on a group of *Virtual CPUs (VCPUs)*. CSP distributes *computation fragments* from a single thread, assigned to a single VCPU, onto multiple cores or *Physical CPUs (PCPUs)*, collocating fragments from multiple VCPUs onto the same core, based on the similarity and dissimilarity of the fragments. In particular, Chakraborty, et al., propose to separate the execution of unprivileged user code from the privileged OS calls it makes. Thus, user code from all VCPUs is executed on a subset of the cores (PCPUs), and OS code from all VCPUs is executed on other cores.

While conceptually appealing, CSP suffers from a fundamental load balance limitation that was mentioned, but not fully addressed, by Chakraborty, et al. When CSP reassigns a computation fragment to execute on a different core (or hardware context), it can lead to contention when another fragment is already assigned to that core. When processing cores are fully utilized by the application, such contention occurs frequently, forcing the hardware/firmware to stall, or *pause*, the execution of the VCPU it is reassigning.

Pausing a VCPU can lead to two different problems: (a) the available hardware resources are under-utilized, as PCPUs recently vacated by paused VCPUs will be left idle, and (b) synchronization overhead in the system software due to locks held by, and cross-calls directed to, paused VCPUs [42]. Both of these problems lead to a performance degradation in a realistic implementation of CSP (20-45% increase in runtime for our workloads). In their initial study, the authors avoid these negative consequences by optimistically assuming a large number of available hardware contexts.

3.1.2 Employing CSP in an OPMS

In an OPMS, the load balance limitation of CSP is dramatically reduced. For example, when the pool of physical cores exceed the number of processors exposed to the OS, the reassignment policy has a larger pool of available cores on which to schedule the collective computation. Thus the probability of resource contention, and the subsequent need to pause a VCPU, is mostly eliminated. Of course, the additional resources (cores) will not be fully utilized, however, only a subset are actually available for simultaneous use anyway due to the SAF constraint. Since fewer VCPUs are paused, the overhead of OS synchronization is also nearly eliminated. For the rest of this paper, we refer an OPMS design employing CSP as *CSP_O*.

For the purpose of comparison, we also evaluate a modified version of Heat and Run on an OPMS (HRTM). We now discuss the implementation of the OPMS we model, as well as specific details of CSP_O and HRTM.

4. IMPLEMENTATION

4.1 Virtualization Support

We assume that the core resources in an OPMS are managed by a lightweight virtualization layer (VMM) implemented in the hardware/firmware. This firmware/hardware, implemented using programmable components, is responsible for the task reassignment and the transfer of computation between different cores. Unlike conventional VMMs designed to support multiple guest OSs, this lightweight VMM does not require the virtualization of memory, I/O devices, privileged instruction, or additional security measures. As the specific functions required for our schemes are fairly simple, without any loss of generality, we have assumed a hardware/firmware implementation. An alternate implementation may even incorporate these functionality in conventional VMMs, which are common place in various platforms.

We use a simple hardware mechanism for *Computation Transfer*, which is the transfer of the register state of a running software thread (VCPU) between on-chip PCPUs. The lightweight VMM uses a section of the physical memory for saving and restoring the register state, and employs the on-chip coherence protocol to communicate this state between on-chip cores. In the UltraSPARC II-Icu architecture we model, this register state, including the entire register window, is about 2.2K data [8]. The state is saved and restored in a sequential fashion using a series of store and load instructions, respectively. The mechanism is similar in spirit to several current microprocessor implementations [1, 12, 41], as well as that proposed in [8]. In our evaluation, we faithfully model the overhead of the computation transfer (latency, bandwidth, cache space and the *energy overhead*).

4.2 Leakage Control in an OPMS

A key implementation challenge for an OPMS design is leakage, which is already a serious concern in current microprocessors, accounting for 30-40% of the total chip power [12, 34]. A large fraction of the leakage is often contributed from the processor cores [34]. Since a typical OPMS design is likely to increase the number of cores in a chip, in a naive design, the total power consumption may actually increase due to leakage from the cores. Fortunately, several circuit techniques have been studied for active leakage control, which are effective at reducing leakage in circuit components with predictable idle periods [40].

For the purpose of this paper, we assume that the processor cores in an OPMS incorporate aggressive leakage control techniques (such as sleep transistors, reverse body bias and so forth) in all the major functional blocks (e.g., register files, instruction window, functional units). Since the OS and user computation are interleaved [8] at relatively coarse granularities (one to two orders of magnitude higher than the typical *breakeven points* reported in

the sleep transistor designs [31, 40] and analytic models [10, 15]), we expect that the transition energy is amortized over the duration of the idle period [10]. However, for a conservative energy estimation, we assume that 2% of the active leakage energy is expended in the processor cores (except the L1 caches and branch predictors which retain their state) not involved in any active computation [10]. When the computation on a core is temporarily suspended, the core's L1 caches are put in a drowsy mode where they expend 20% of the active leakage energy [11].

4.3 CSP_O: Core Provisioning and Scheduling

The *CSP_O* models an OPMS that employs CSP for assigning computation from the eight VCPUs across 12 cores (see Section 5.1 for our baseline multicore and workload configurations). For the OS-intensive commercial workloads we study in this paper, we statically provision a certain number of cores for executing OS code, and certain number of cores for user code, based on the fraction of time each workload spends in OS or user mode. In a realistic implementation, we envision this provisioning could easily be performed at runtime by simply measuring the amount of OS and user execution. We do not consider short-lived register window traps for computation transfer. Other OS computation (e.g., system calls, page faults, interrupt handling) trigger a computation transfer, and are carried out in the cores provisioned for OS computation. For Apache and Zeus, eight of the 12 cores are used for OS computation, while the rest are provisioned for user computation. The rest of the workloads employ the opposite provisioning of eight user and four OS cores. We maintain a static mapping of OS and user computation from a VCPU to their corresponding cores, similar to the *Thread Assignment Policy* in [8]. However, unlike the optimistic methodology adopted in [8], when a VCPU attempts to use a core that is already being used by another VCPU, we *do* pause the execution of the former VCPU until the core becomes available. This implementation allows us to evaluate CSP_O in a more realistic system design.

The scheduling employed in CSP_O is implemented using a simple map, and a small wait queue of VCPUs at each PCPU. For each VCPU, the map maintains two PCPU mappings: one for user computation and the other for OS computation. This map is kept with each core, and setup once by the firmware based on the workload profile as described above. The control overhead, though not modeled explicitly, can be overlapped by the CT as the saving and restoring of register state is sequential. Therefore, during a CT, the reassigned PCPU for a VCPU is not required till its state is saved in the caches—allowing sufficient time to: (1) determine the next VCPU to execute on the current PCPU (from the wait queue); and (2) determine which PCPU to resume (from the map) the computation of the vacating VCPU and communicate to the new on-chip PCPU. Both the map and the wait queue can be easily implemented in the hardware allowing faster operation, or in a conventional full feature VMM implementation to offer more flexibility. We also expect this particular flavor of dynamic task reassignment to be fairly scalable to higher number of cores, though a detailed scalability study is beyond the scope of this paper. In a hardware implementation, both these structures will have negligible energy overhead due to infrequent access (once in every 5–20 thousand instructions).

4.4 HRTM

HRTM is a modified version of the Heat-and-Run thread migration proposed by Gomaa, et al. [14]. In HRTM, we use computation transfer, instead of OS scheduling, to periodically transfer a virtual CPU among multiple cores. Each VCPU spends at least 8 ms on a processing core before it is transferred, so as to amortize the negative caching effects. The idle period in each core lasts for at least 4 ms to allow sufficient time for cooling critical hotspots. As the RC time constants for individual hotspots are much smaller than that of the entire processor, we find that this idle time is sufficient to significantly reduce the temperature of the issue queue, the register file, and so forth, before another computation is resumed.

Technology Node	45nm at 3.00 GHZ and 1.0V
Fetch/issue/commit	4 instructions / cycle
Integer pipeline	12 stages
I-Window & ROB	128 entries, OOO issue
Load & store queues	32 entries each, w/ bypassing
Store buffer	32 entries, processor consistency
YAGS branch pred.	4k choice, 1k except, 6 tag bits
Priv. L1 instr. cache	64kB, 8-way, 2-cycle, coherent
Priv. L1 data cache	64kB, 8-way, 2-cycle, write-back
On-chip shared L2 cache	16MB, 16 banks, 32-way, 60-cycle load to use, pipelined, inclusive
Coherence Protocol	Directory based MESI
On-chip network	Point-to-point, 25-cycle
Main Memory	255 cycle load-to-use, 40GB/sec

Baseline	OPMS Configurations
8 cores	12 cores
16MB, 16 bank L2	12MB, 12 bank L2

Table 1: Baseline processor parameters

5. EXPERIMENTAL METHODOLOGY

5.1 Multicore Systems

We use full system, cycle-accurate simulation based on SIMICS micro-architecture interface (MAI), which can boot an unmodified Solaris operating system and execute commercial workloads. Our simulation infrastructure models the functional aspects of UltraSPARC IIICu CPUs, which implement the SPARC V9 ISA. However, we use our timing model to enforce the timing characteristics of a multicore system built with out-of-order processor cores. Various parameters of this system are given in Table 1, while our workloads are described in Table 2. In addition, we model a hardware filled TLB, with negligible penalty, so as to avoid a high overhead from the software filled TLB in the SPARC V9 ISA.

All the benchmarks, setup on Solaris 9, are initially run for a long duration (up to several simulated minutes) to warm up the applications and OS disk caches, and we then collect memory traces for several simulated seconds to warm up the large L2 cache before starting timing runs. For timing runs, we use 1300 transactions in Apache and Zeus, and the rest are run for 100 million committed user instructions. Due to the inherent variability [2], we add a small random variation to the main memory latency, and run several trials of each benchmark per experiment. We present the average results, and the 95% confidence interval wherever applicable.

We model a point-to-point logical interconnect where each node of the network is either a processor core or an L2 cache bank of 1MB. In all, there are 24 nodes in the network (fixed across the multicore configurations modeled). In the baseline system, these nodes are occupied by eight processor cores and 16 banks of the L2 cache. In each of the OPMS configurations, consisting of 12 cores, we remove four banks of the L2 cache and replace them with cores. Therefore, the aggregate L2 cache size in the OPMS is reduced to

Apache	We use the Surge client to drive the open-source Apache web server, version 2.0.48. We do not use any think time in the Surge client to reduce OS idle time. Both client and server are running on the same, 8-processor machine.
OLTP	OLTP uses the IBM DB2 database to run queries from TPC-C. The database is scaled down from TPC-C specification to about 800MB and runs 192 concurrent user threads with no think time.
pgoltp	pgoltp also runs queries from TPC-C, but uses the PostgreSQL 8.1.3 database [32] driven by OSDL's DBT-2 [30]. Unlike IBM's DB2, PostgreSQL performs I/O through the OS's standard interfaces and utilizes the OS's disk cache.
Zeus	We use the Surge client again to drive the Zeus web server, configured similarly to Apache.
pmake	Parallel compile of PostgreSQL using GNU make with the `-j 64` flag using the Sun Forte Developer 7 C compiler. Unlike the server workloads, pmake consists of multiple processes running in separate virtual address spaces.

Table 2: Workloads

accommodate additional processor cores. The area overhead from the additional cores and interconnect is compensated by the cache banks they replace [16]. We also assume that the target power envelope allows up to eight simultaneously active processing cores as described in the Table 1, and simultaneous execution on all the processor cores in the OPMS is inadmissible. Consequently, in all of our workloads, we expose eight virtual processors (VCPUs) on which the OS schedules software threads to run. The OS and application configuration is identical across all the multicore systems we evaluate.

5.2 Energy Estimation

To estimate power and energy consumption, we have integrated a combination of tools in our simulation infrastructure. Processor core activity and power is largely derived using the circuit models from Wattch [7]. The memory hierarchy uses CACTI 4.2 to model dynamic energy dissipation and leakage in the caches [39]. Since the L2 cache is considerably large, and may account for an unacceptable amount of static energy consumption [28], we assume it is implemented with stacked devices. This implementation can effectively reduce the leakage by 3X, providing a much more energy efficient design point [35].

Calibrating Wattch.

Analytic power modeling tools such as *Wattch* rely on tracking cycle-by-cycle access history of various functional blocks in a processor core, and on certain assumptions about circuit design styles. While such tools are considered reasonable at estimating the power/energy consumption *trends*, several previous studies indicate their inaccuracy in estimating the *absolute* power [23, 25]. To overcome this drawback, we use a combination of other tools to calibrate the energy consumption reported by the power model.

First, we derive the total power consumption (both static and dynamic) of major functional blocks of a core when they are operating

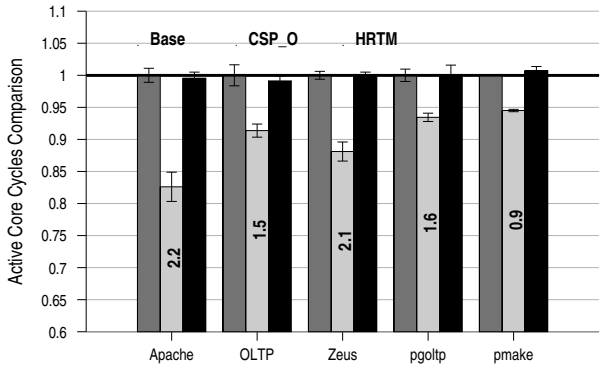

Figure 3: Relative cycles of activity. The labels on bars for CSP_O indicate the cycles overhead (percentage) for CT.

close to their TDP (thermal design points) using HotSpot [37]. We use the *hotfloorplan* tool (included in the HotSpot tool set) to obtain the floorplan of a processor core in the 45nm technology node. For the purposes of this calibration, we focus on a single core because: (a) accurate floor planning of a chip with eight or more cores is beyond the scope of this paper and (b) the effect of lateral heat diffusion is mostly localized [37], and therefore the temperature of each functional block is primarily dependent on the nearby functional blocks of the same processing core. After obtaining the total power, we estimate the static power as a temperature-dependent fraction of dynamic power [37], and determine the dynamic power component. Finally, we use this dynamic power to calibrate the dynamic energy consumption reported by the Wattch-based power module, when running a high IPC workload at the nominal voltage-frequency. Note that a similar methodology was adopted by Li and Martinez [25]. For our timing runs, we initialize the temperature of the heat sink, and various functional blocks of the core, by several iterative runs of 500 ms, until there is little variance between two successive runs. During the timing runs, we update the temperature of various functional blocks every 32K cycles, and also adjust the static power component of each block based on its temperature.

6. EXPERIMENTAL RESULTS

We now present the results of a quantitative analysis of the energy benefits of CSP_O, our proposed scheme for using an OPMS. From our experiments, we wish to answer several questions: How much energy efficiency in each core is obtained through CSP_O? What are the thermal advantages of CSP_O? What is the effect on cache energy? What is the overall energy-delay product? Answers to these questions are presented in the following sections. We present the average results, and the 95% confidence interval, wherever applicable.

For purposes of comparison, we also present quantitative results from the HRTM scheme we described in Section 4.4. The HRTM scheme we model, spreads the computation from eight VCPUs among 12 cores without any attempt to specialize the cores in the process. Since inactive cores, after enabling the sleep signals, consume little static or dynamic power, to the first order, we would expect HRTM to perform similar to the baseline in various aspects, except improving upon the thermal characteristics of the cores.

6.1 Compute Efficiency

When performing CSP_O, the dynamic specialization of each core's predictive structures improve its *compute efficiency*: the ef-

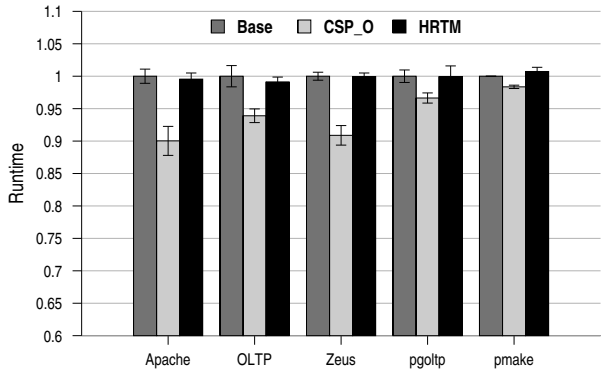

Figure 4: Relative runtime. The CT overhead (see Figure 3) for CSP_O and HRTM is included in the runtime.

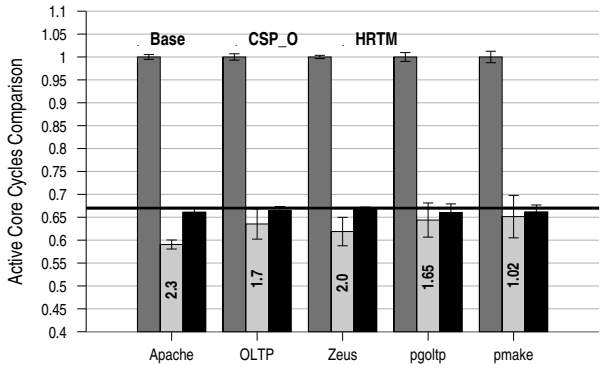

Figure 5: Comparison of energy consumption per core. The horizontal line at 0.67 shows the expected (per core) energy reduction from using 12 cores instead of 8. Labels on the CSP_O bars indicate the per core energy overhead (percentage) from CT.

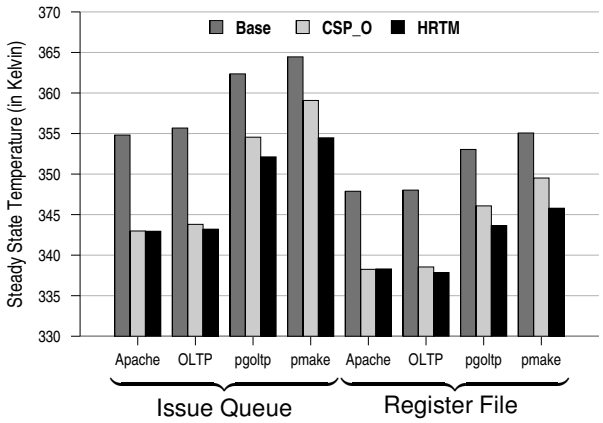

Figure 6: Steady State Thermal Characteristics

ficiency of performing an assigned computation. The boost in the compute efficiency causes an increase in IPC, and a commensurate reduction in the amount of time cores are actively engaged in computation for the same amount of work (workload transactions). We show a comparison of aggregate active core cycles in Figure 3. Across all workloads, the active core cycles are reduced by 5–17%, showing a substantial improvement of the compute efficiency in CSP_O cores. The active core cycles shown in the figure *also* include the overhead of computation transfer (CT), which varies from 0.9% in *pmake* to 2.2% in Apache for CSP_O. The overheads are much smaller in HRTM due to infrequent CT.

Due to the resource contention described in Section 3.1.1, which is not modeled by Chakraborty, et al. [8], the decrease in active cycles per core does not directly translate to an improved runtime, as shown in 4. Instead, CSP_O does not keep eight cores active all the time. For example, eight cores are employed simultaneously only 48% and 78% of the simulated cycles in Apache and OLTP, respectively (data not shown for brevity). Despite this lower core utilization, we still observe a modest reduction in the runtime ranging from 2–10%.

The CT overhead in our experiments is fairly low, allowing schemes like CSP_O to be effective. This low overhead is primarily due to two factors. First, though CTs are much more frequent than OS timer interrupts, computation fragments between two CT events are substantial in size. On average, these events are triggered every 5K instructions in Apache (most frequent), and every 20K instructions in pmake (most infrequent), similar to those reported in [8]. Second, current multicore systems allow fast and high bandwidth communication between on-chip cores. Using the existing on-chip caches and coherence protocol, we find that the data transfer latency of a single CT event is on the order of a few hundred cycles. For HRTM, the CT overhead is negligible due to their low frequency.

6.2 Core Energy

In Figure 5, we show the average energy expenditure of each processor core in the baseline and the two OPMS schemes. This figure only shows the energy from the core-logic: energy from private caches is shown later in Figure 7. Because the OPMS configurations are over-provisioned, we expect the average energy of each core to be 8/12 (or 0.67) of the baseline, since they are inactive roughly one-third of the time. Indeed, this is the case for HRTM since it does not perform any specialization of the cores. For CSP_O, however, the energy per core is significantly lower than the degree of over-provisioning for some benchmarks due to reduction in active cycles shown in Figure 3.

The actual savings also reflect how effective CSP_O is in specializing the cores for various workloads: *pmake*, for example, shows limited energy benefit beyond that of the over-provisioning, as separating the user and OS computation is not as effective for this workload as compared to the rest. This is due to much longer phases for OS and user computation [8], and inherently better locality of references, offering smaller room for improvement.

6.3 Thermal Characteristics

In Figure 6, we show the steady state temperature of two typical hotspots in a processor core: the issue queue and the register file. For each benchmark, three sets of bars, from left to right, show the temperature of the baseline, CSP_O, and HRTM, respectively. These steady state temperatures are obtained from the power trace of much longer runs (see Section 5). CSP_O is able to significantly reduce the temperature in Apache, OLTP, and Zeus (data for Zeus is omitted, but it is similar to Apache). In OLTP, for example, temperatures of the issue queue and register file are reduced by 11 and 10 degrees, respectively. CSP_O is less effective in reducing the temperature in *pmake* as the utilization of various cores varies significantly. In particular, we find that two cores, provisioned for

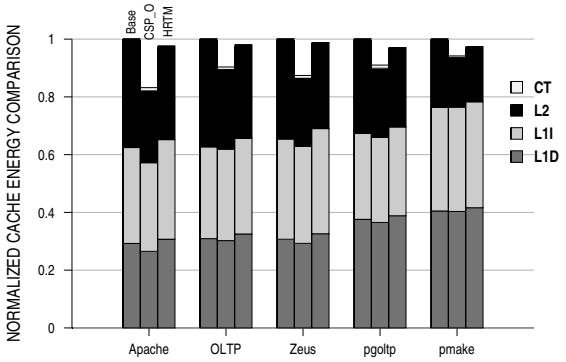

Figure 7: Comparison of energy consumption in caches.

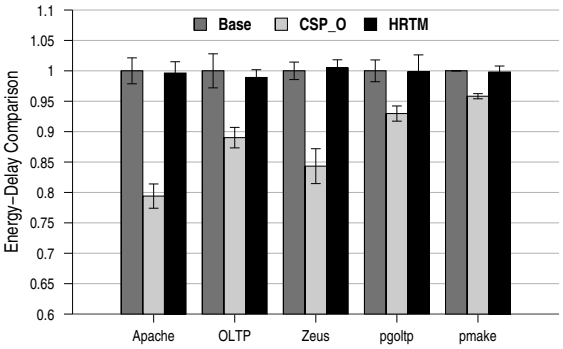

Figure 8: Energy delay product comparison.

OS computation, see much less use due to a significantly lower OS computation from their corresponding VCPUs. We observe that HRTM is more effective at reducing the temperature, particularly in *pmake*, as it more evenly spreads computation among the cores. The temperatures observed by *pmake* and *pgoltp* are significantly higher than that of *Apache* and *OLTP* due to the higher IPCs in these two applications.

In addition to the steady-state temperatures, we have also examined the temporal variance in the transient temperature of each core, and the spatial variance in the temperature among different cores. In the baseline system, both the temporal and spatial variance is low due to the homogeneous nature of these workloads, as well as relatively small IPC variations and short phases. CSP_O similarly exhibits a low temporal variance (marginally higher than the baseline), as task are migrated among cores frequently (every 5–20 thousand instructions). The spatial variance is also small, since all cores are fairly evenly utilized (except for the two OS cores in *pmake*). Again, this spatial variation is slightly higher than the baseline due to the nature of OS and user computation in each workload. In contrast to the baseline and CSP_O, HRTM shows much larger temporal variance, as the periods of active computation are interleaved at a much longer time-scale. This results in long periods of high heat and temperature (often similar to the baseline) and long periods of cool down (much lower than average temperature). However, spatial variation is minimal, as by design, HRTM spreads the entire computation evenly among the available cores.

6.4 Cache Energy

In Figure 7, we show the comparison of energy expended in the memory hierarchy. For each workload, the three bars (from left to right) show the results for the baseline, CSP_O, and HRTM, respectively. Each bar is further broken down into four parts showing the energy consumption in three caches: L1 data cache (bottom), L1 Instruction cache (middle), L2 cache (top) and energy overhead due to CT (CT). The energy from the L1 caches is increased in several OPMS configurations, especially for HRTM due to the increase in leakage from a larger aggregate L1 capacity. However, Apache and Zeus in CSP_O show a slight improvement in L1 energy largely due to the significant improvement in their runtime (10% and 9%, respectively), which balances the increase in leakage energy from more L1 caches. The L1 energy is also reduced by lowering branch mis-speculations in CSP_O, which reduces accesses into the L1.

In contrast, we see a reduction in L2 energy in all OPMS configurations. In HRTM, this is solely due to the reduction of leakage energy from a smaller L2 cache. CSP_O shows much larger reduction over HRTM, especially for Apache, OLTP, and Zeus. Apache, for example, shows an overall cache energy reduction of 18% compared to the baseline. This additional reduction in CSP_O is derived primarily from a significant reduction in the L2 accesses, especially for instructions, resulting in savings in switching energy. Of course, the leakage energy is also conserved, due to both an improved runtime and a smaller L2.

The energy overhead for data transfers due to CT is also marginal in CSP_O (up to 1.1% in Apache), as we expect from its low latency overhead in Figure 3. As explained in Section 4.1, the CT mechanism has two parts: saving VCPU registers in the current PCPU and restoring the same at the remote PCPU. The first part mostly requires local private cache access, especially for Apache and Zeus where CT events are more frequent and thus the regions of memory used for this purpose remain in the private caches, thereby expending low energy. A substantial component of the energy overhead comes from the second part, which requires access into the L2 cache, interconnect and the remote cache. The CT overhead reported here is the directly measurable overhead: switching energy due to register state transfer. Energy overhead from other thread data, which are transferred on demand, are not included in the CT overhead explicitly, but they appear in various components of cache energy shown in the figure.

The multicore systems we model use a directory based coherence protocol, which does not need to broadcast all messages over the interconnect. While our simulator infrastructure currently lacks a model for global interconnect energy estimation, we expect that CSP_O will perform favorably compared to the baseline system and HRTM. First, we find that the aggregate bandwidth requirement (which directly translates into interconnect switching activity) is improved even after accounting for the computation transfer overhead. Second, the leakage component of the interconnect energy is also favorable due to a significant reduction in runtime for CSP_O.

6.5 Energy Delay Product

We show the comparison of the energy-delay product across the three schemes in Figure 8. As expected, CSP_O performs significantly better than both the baseline machine and HRTM. This energy-delay benefit is derived both from the conservation of overall energy, as well as the improvement in runtime. Quantitatively, the energy-delay improvement tracks well with the runtime reduction observed: more improvement is seen in benchmarks which also show higher speedup, indicating efficient use of microarchitectural structures in CSP_O. The best improvement is achieved by Apache, resulting in a 20% lower energy-delay product. Zeus and

OLTP also show impressive energy-delay improvements of 15% and 10%, respectively. On the other hand, *pgoltp* and *pmake* show more modest savings, also reflected in their smaller runtime improvement. HRTM, on the other hand, performs similarly to the baseline system.

In summary, CSP_O is able to provide both superior energy *and* delay, by efficiently employing a larger pool of core resources, while remaining within the same power envelope, and SAF, compared to the baseline and HRTM system. Energy and delay benefits are derived from several components of a multicore system, clearly demonstrating the effectiveness of the novel OPMS design paradigm.

7. RELATED WORK

While previous research has implicitly used an over-provisioned system to improve energy efficiency and thermal characteristics [14, 24], this work presents a novel perspective on designing an over-provisioned system in the light of current technology trends. We leverage the Computation Spreading work by Chakraborty, et. al. [8], to impart dynamic specialization on structurally identical cores of an OPMS.

Several previous proposals have looked into the power/performance implications of varying resource provisioning for caches and cores using multi-programmed workloads (e.g., [26]) and multithreaded scientific workloads (e.g., [28]). In this work, we focus on multithreaded server class workloads and evaluate the energy-efficiency of an OPMS system. To the best of our knowledge, no previous work has looked at utilizing an over-provisioned system for these workloads.

Many past proposals advocate activity migration to mitigate thermal hot spots [14, 27, 37]. While adopting different mechanisms and underlying microarchitectures, all of them attempt to take rectifying action after a certain threshold temperature is sensed, indicating a forthcoming thermal emergency. In this work, we continually move the computation around to avoid exercising one particular resource for a long enough duration to incur a thermal emergency.

For memory-bound applications (such as server applications), prior research has demonstrated that voltage-frequency scaling yields a superior energy-delay product, as long as the performance degradation is within an acceptable limit [17, 26]. The bulk of the energy benefit is derived from supply voltage scaling, while the energy-delay improvement arises due to only limited increases in delay, since memory latencies are not scaled in tandem with frequency. However, aggressive scaling of voltage may not be feasible in the future technology generations, where the expected supply voltage will be below 1.0V [9, 13, 14]. Moreover, *dynamic* frequency scaling of individual cores is less attractive when running multithreaded server workloads: First, poor performance and unpredictable behavior can result from these complex applications in the presence of performance asymmetry [4]. Second, multithreaded server application have much more commonality in the code executed by multiple threads [8], and therefore show less variability in the memory stall times experienced by individual cores, than do multi-programmed SPEC workloads (as used in [17]). For these reasons, we did not consider the impact of dynamic voltage and frequency scaling in this paper. Instead, we assumed a slightly reduced voltage and much reduced frequency, modeling multicore systems that already incorporate significant benefit from static or dynamic VFS techniques

8. CONCLUSIONS

This paper considered a novel proposal for future multicore processors where, despite an increasing number of transistors, power considerations will significantly limit the number of chip resources that can be active simultaneously. Using the projected device characteristics from the ITRS Roadmap, we showed that the *Simultaneously Active Fraction (SAF)*—the fraction of resources that can be used simultaneously to perform computation—is falling with each successive technology generation.

To use the growing pool of resources, while honoring the constraints imposed by a shrinking SAF, we proposed the use of a SAF-aware Over-provisioned Multicore System (OPMS) design. Unlike traditional systems, where processing core resources are provisioned for simultaneous and continuous use, in an OPMS the number of processing cores on the chip exceeds the allowable number of simultaneously active cores. To remain within the allowable power budget, individual cores transition between active and inactive states at different times, resulting in a system with a constantly-changing set of processing resources.

We considered how to use an OMPS to improve the computation and energy efficiency while processing multithreaded server workloads. Leveraging a recently-proposed technique *Computation Spreading* [8], and a lightweight VMM, to distribute the collective computation of multithreaded workloads across all of the cores in an OPMS, we saw how the prediction structures of a given core could be specialized, thus improving its computation and energy efficiency. Using an elaborate evaluation infrastructure, we observed an energy-delay product improvement of 5–20% for our multithreaded, server class applications. In addition, regular interleaving of computation with idle periods naturally improved the chip's thermal characteristics.

This paper is an initial foray into what we feel will be a prominent model for processing chips in the future: chips with more processing cores than can be active at any given time. We expect this to be the case since the continuing demand to reduce the SAF will move the granularity for activity reduction from small pieces of logic (e.g., via clock gating) to functional units or cache blocks, to entire processing cores. We considered one scenario for such a system and showed how it could be used to achieve computing and energy efficiency for multithreaded server workloads. Many other opportunities are likely to arise with such over-provisioned multicore systems, and exploiting these opportunities will create many new problems that will have to be solved not only in the hardware and firmware, but even in the software layers above.

9. ACKNOWLEDGEMENTS

This work is supported in part by National Science Foundation (NSF) grants CCF-0702313 and CNS-0551401, funds from the John P. Morgridge Chair in Computer Sciences and the University of Wisconsin Graduate School. Sohi has a significant financial interest in Sun Microsystems. The views expressed herein are not necessarily those of the NSF, Sun Microsystems or the University of Wisconsin.

10. REFERENCES

[1] Advanced Micro Devices. *AMD64 Architecture Programmer's Manual Vol. 2: System Programming*, Dec 2005.

[2] A. R. Alameldeen and D. A. Wood. Variability in architectural simulations of multi-threaded workloads. 2003.

[3] H. Ananthan and K. Roy. A fully physical model for leakage distribution under process variations in nanoscale double-gate cmos. In *Proc. of 43th DAC*, 2006.

[4] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *Proc. of 32nd ISCA*, 2005.

[5] B. M. Beckmann. *Managing Wire Delay in Chip Multiprocessor Caches*. PhD thesis, University of Wisconsin-Madison, 2006.

[6] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4), 1999.

[7] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proc. of 27th ISCA*, 2000.

[8] K. Chakraborty, P. Wells, and G. Sohi. Computation spreading: Employing hardware migration to specialize CMP cores on-the-fly. In *Proc. of 12th ASPLOS*, 2006.

[9] B. Dennington. Low power design - from technology challenges to great products: Keynote. In *Proc. of ISLPED*, 2006.

[10] S. Dropsho, V. Kursun, D. H. Albonesi, S. Dwarkadas, and E. G. Friedman. Managing static leakage energy in microprocessor functional units. In *Proc. of 35th MICRO*, 2002.

[11] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Proc. of 29th ISCA*, 2002.

[12] J. Fredrich et al. Design of the Power6 microprocessor. In *Proc. of Intnt'l Solid-State Circuits Conf.*, 2007.

[13] K. Ghose, P. Bose, and K. Skadron. Computer architecture research directions: Panel discussion on low power design and temperature management. Workshop in HPCA, 2007.

[14] M. Gomaa, M. D. Powell, and T. N. Vijaykumar. Heat-and-run: leveraging SMT and CMP to manage power density through the operating system. In *Proc. of 11th ASPLOS*, 2004.

[15] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose. Microarchitectural techniques for power gating of execution units. In *Proc. of ISLPED*, 2004.

[16] J. Huh, D. Burger, and S. W. Keckler. Exploring the design space of future cmps. In *Proc. of 10th PACT*, 2001.

[17] 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 *Proc. of 39th MICRO*, 2006.

[18] ITRS. Executive summary. *Intnt'l Technology Roadmap For Semiconductors*, 2005.

[19] ITRS. Process integration, devices and structures. *Intnt'l Technology Roadmap For Semiconductors*, 2005.

[20] ITRS. Systems drivers. *Intnt'l Technology Roadmap For Semiconductors*, 2005.

[21] ITRS Working Group Models: Process Integration Chapter. Mastar: Model for assessment of cmos technologies and roadmap. http://www.itrs.net/models.html.

[22] A. Jaleel, M. Mattina, and B. Jacob. Last level cache (LLC) performance of data mining workloads on a CMP âĂŤ a case study of parallel bioinformatics workloads. In *Proc. of 12th HPCA*, 2006.

[23] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *Proc. of 36th MICRO*, 2003.

[24] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *Proc. of 31st ISCA*, 2004.

[25] J. Li and J. F. Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *Proc. of 12th HPCA*, 2006.

[26] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron. CMP design space exploration subject to physical constraint. In *Proc. of 12th HPCA*, 2006.

[27] W. Liao, L. He, and K. Lepak. Temperature and supply voltage aware performance and power modeling at the microarchitecture level. *Trans. on Comp.-Aided Design of Integrated Circuits and Sys.*, 24(7), 2005.

[28] M. Monchiero, R. Canal, and A. González. Design space exploration for multicore architectures: A power/performance/thermal view. In *Proc. of 20th ICS*, 2006.

[29] E. J. Nowak et al. Scaling beyond the 65 nm node with FinFET-DGCMOS. In *Proc. of the Custom Integrated Circuits Conf.*, 2003.

[30] Open Source Development Labs. Database test suite. http://osdldbt.sourceforge.net/. Viewed 5/29/2007.

[31] E. Pakbaznia, F. Fallah, and M. Pedram. Charge recycling in MTCMOS circuits: concept and analysis. In *Proc. of 43th DAC*, 2006.

[32] PostgreSQL. PostgreSQL: the world's most advanced open source database. http://www.postgresql.org/.

[33] P. Ranganathan and N. Jouppi. Enterprise IT trends and implications on system architecture research. In *Proc. of 11th HPCA*, 2005.

[34] S. Rusu et al. A 65-nm dual-core multithreaded XeonÂő processor with 16-MB L3 cache. *IEEE Solid-state Circuits*, 2007.

[35] N. Sakran, M. Yuffe, M. Mehalel, J. Doweck, E. Knoll, and A. Kovacs. The implementation of the 65nm dual-core 64b merom processor. In *IEEE International Solid-State Circuits Conference*, 2007.

[36] X. Shi, F. Su, J. kwon Peir, Y. Xia, and Z. Yang. CMP cache performance projection: Accessibility vs. capacity. In *Workshop on Design, Architecture and Simulation of Chip Multi-Processors*, 2006.

[37] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture: Extended discussion and results. Technical Report CS-2003-08, University of Virginia, 2003.

[38] J. Stoess, C. Lang, and F. Bellosa. Energy management for hypervisor-based virtual machines. In *Proc. of 2007USENIX Technical Conference*, 2007.

[39] D. Tarjan, S. Thoziyoor, and N. Jouppi. Cacti 4.0. Technical Report 20060606, HP Labs, 2006.

[40] J. W. Tschanz, S. G. Narendra, Y. Ye, B. A. Bloechel, S. Borkar, and V. De. Dynamic sleep transistor and body bias for active leakage power control of microprocessors. *IEEE Solid State Circuits*, 38(11), 2003.

[41] R. Uhlig et al. Intel virtualization technology. *Computer*, 38(5), 2005.

[42] P. Wells, K. Chakraborty, and G. Sohi. Hardware support for spin management in overcommitted virtual machines. In *Proc. of 15th PACT*, 2006.

[43] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Ecosystem: managing energy as a first class operating system resource. In *Proc. of 10th ASPLOS*, 2002.