# An Evaluation of Server Consolidation Workloads for Multi-Core Designs

‡Natalie Enright Jerger, *Dana Vantrease, and ‡Mikko Lipasti
‡Electrical and Computer Engineering Department, University of Wisconsin-Madison
*Computer Science Department, University of Wisconsin-Madison
enrightn@cae.wisc.edu, danav@cs.wisc.edu, mikko@engr.wisc.edu

*Abstract*— **While chip multiprocessors with ten or more cores will be feasible within a few years, the search for applications that fully exploit their attributes continues. In the meantime, one sure-fire application for such machines will be to serve as consolidation platforms for sets of workloads that previously occupied multiple discrete systems. Such** *server consolidation* **scenarios will simplify system administration and lead to savings in power, cost, and physical infrastructure. This paper studies the behavior of server consolidation workloads, focusing particularly on sharing of caches across a variety of configurations. Noteworthy interactions emerge within a workload, and notably across workloads, when multiple server workloads are scheduled on the same chip. These workloads present an interesting design point and will help designers better evaluate trade-offs as we push forward into the many-core era.**

## I. Introduction

As device counts continue to increase, multi-core chips with 10s to 100s of cores are fast approaching on the horizon. Sun Microsystem's Niagara offers 8 cores and 32 threads [23]. Intel's Tera-scale computing project [18] promises to offer 80 cores on a single die. In addition to the research and design challenges faced by these large-scale multi-core designs, the industry faces the challenge of finding applications and workloads to benchmark and evaluate these designs. Exploring server consolidation workloads is one answer to this growing problem.

Traditionally, companies would pick a server platform for a particular application, and then install one or more physical systems to serve the computational needs of that application. As the computational needs increased, perhaps as the company grew in size, it purchased more servers to keep up with its growth and to support new solutions. The sheer abundance of servers resulting from this practice has today evolved into a nuisance, in terms of both cost and system administration, and has contributed to the coining of the term "server sprawl." Across servers, server sprawl results in servers taking up more floor space, consuming more electricity, and being less reliable than their consolidated server counterparts. Meanwhile, a single server may actually be underutilized and yet limit application scalability due to system-specific bottlenecks.

Fortunately, the resurgence of virtualization technology [13, 32] has helped to alleviate many of the problems by consolidating several physical servers on one single physical high-end server. In the process, each physical server's applications and operating system are migrated to a central server where the applications coexist on virtual machines that are guest hosted on a virtualization platform. The "virtual servers" potentially share the entirety of available physical resources but are given the impression that they are running in isolation. By centralizing the applications and sharing the physical resources, management costs are reduced. Amongst many other benefits, servers can experience higher utilization and may dynamically meet changing demands by scaling their resource requirements [21]. Section II lists the consolidated applications under evaluation while more remarks on how the applications are run and evaluated can be found in Section IV.

Multi-core architectures, which have emerged in a performance-driven effort to provide more on-chip parallelism, are well-suited for commercial workloads running in a server consolidation environment. Each workload may be given the impression of running on its own private system, but in actuality, the hardware may be shared with other workloads in a variety of possible arrangements. Figure 1 illustrates two different arrangements of four workloads on a server consolidation system with groups of four cores sharing a last level cache. This figure shows four distinct workloads, each consisting of four threads, running on a 16 core CMP with two different scheduling policies. Round robin and affinity configurations are shown and will be discussed in further detail in Section III-C.

These workloads, which tend to be multi-threaded and communication-intensive applications, are a good match with the multi-core architecture's on-chip memory system, which allows for fast inter-thread communication while the multiple cores allow for concurrency. As feature sizes continue to shrink, there is an opportunity to populate the chip with even more cores. Additional cores increase the effective processing capacity of the chip, and in turn customers can obtain even higher degrees of consolidation within a single chip.

By integrating a large-scale multiprocessor onto a single-chip, where several of the resources may be shared, interference can come from many sources; this interference occurs at a much finer granularity than in traditional multi-chip, multi-board servers. Many cores may compete for the same interconnect bandwidth, memory controllers, and cache resources. Sharing these resources to provide superior performance as well as ensuring fairness has been the subject of recent work (discussed in Section VI) and will continue to grow in importance. The interconnect, which must be able to provide fast and reliable communication from any one core to any other, may be shared across cores. Memory controllers, which are central arbiters between the chips and

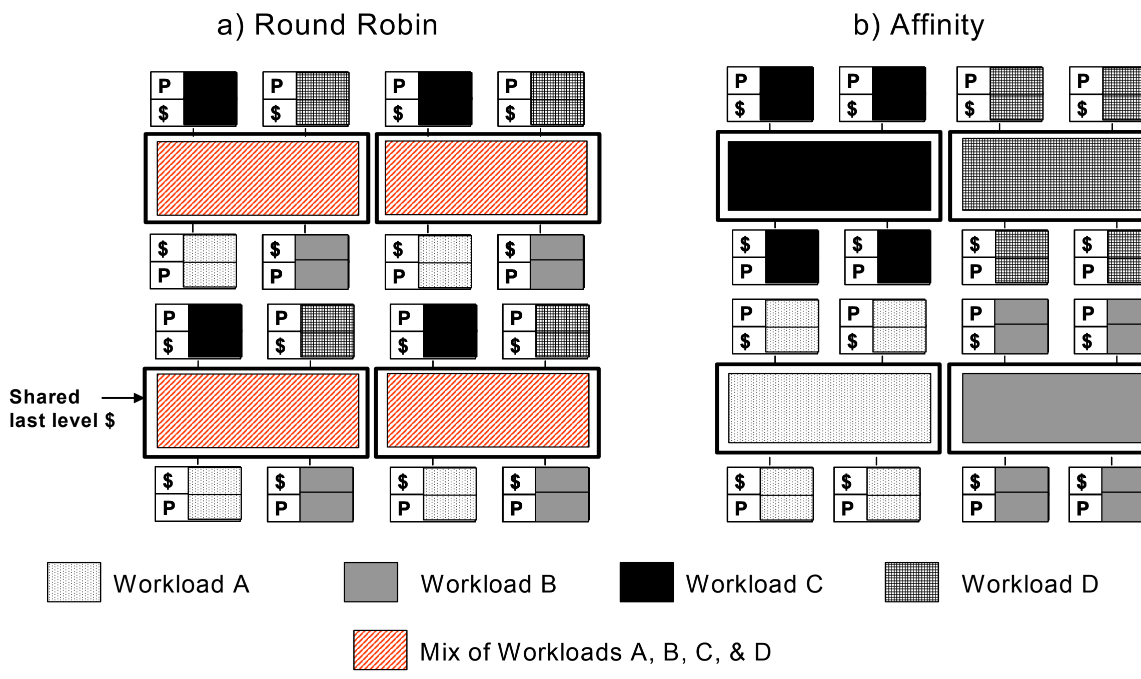

Fig. 1. System running server consolidation workloads

memory, may receive requests from multiple cores. Last, but not least, caches may be shared to varying degrees amongst cores. The last level cache is the final opportunity to keep the request on-chip before incurring the high latency effects of going off-chip. Thus every effort must be made to satisfy the request on-chip.

With the pressure to add more processing cores, cache area has become increasingly precious. In response, many designs dictate cores share a portion, or entirety, of the chip's last-level cache memory. By allowing shared access to the last level cache, the designs strive to increase cache utilization and consequently curtail the occurrence of high-latency performance-damaging off-chip misses. In the ideal scenario, the shared cache adapts to the changing footprint needs of running threads, while cutting down on on-chip coherence traffic and instances of replicated data. In the non-ideal scenario, the workloads stress the cache and cause destructive interference; capacity and conflict misses grow rampant and performance degrades significantly [17]. This destructive interference in the caches can spill over to cause congestion in the interconnect and put additional pressure on the memory controllers and push more data off chip. The trade-offs in the continuum of designs, between private and full-shared caches, in conjunction with the behavior of the programming environment, play a large role in the timely manner in which the request is satisfied and is the focus of our work. The complex relationship between constructive and destructive interference and its effects are key motivators in this study. Details regarding a variety of cache organization are found in Section III. In Section V, we study the interaction at the last level cache.

Simulation methodology has been a topic of significant research for both simultaneous multi-threaded (SMT) designs and for multiprocessor designs [7, 25] The simulation time of large numbers of cores can be prohibitive, especially when combined with a wide variety of workloads combinations that can occur in the server consolidation environment [16]. We discuss our approach to these simulation challenges in Section IV.

We show, across a variety of shared cache configurations, that a commercial workload's memory behavior can be affected in unexpected ways by other workloads. Thus, in Sections VII and VIII, we argue that the characterization we present is insightful to OS/VM developers, code-tuners, architects, and begs further research.

The main contributions of this work are as follows:
1. Explore the importance of server consolidation workloads to a variety of research communities.
2. Identify interactions within server consolidation workloads that will open new avenues of research.
3. Demonstrate the need to evaluate commercial workloads not just in isolation but in a consolidated environment.

## II. Workload Overviews

The workloads presented are all server multi-threaded commercial workloads, representative of workloads that might be found running on a company's consolidated server. They come from two consortia, the Systems Performance Evaluation Cooperative (SPEC) [34] and the Transaction Processing Performance Counsel (TPC) [35].

As previous studies have pointed out, commercial workloads exhibit large degrees of inter-thread sharing [2,12,31]. For instance, in TPC-W, when caches are private to the core, a high percentage of misses result in a cache-to-cache transfers and a significant portion of these misses are serviced from a remote cache line that is in the dirty state [6]. The effects of cache-to-cache transfers in the workloads studied make them interesting subjects, especially from the perspective of cache coherence, protocol design, and cache organization. The organization is in part interesting because cache-to-cache transfers can be streamlined or even avoided completely, at a cost, when appropriate levels of sharing are built into the hierarchy. TPC-H exhibits significant intra-query parallelism through the collaborating operators within each query's execution, while the join/merge activity causes plenty of sharing and synchronization across threads. The descriptions of the particular workloads, SPECjbb, SPECweb, TPC-H, and TPC-W, can be found in Table I.

Table II gives further insight into the individual workloads that are consolidated in this study. The percentage of misses to the last level of private cache that result in an on-chip cache to cache transfer are given as well as the number of cache line sized blocks that are touched during the simulation. These workloads exhibit a range of misses that are satisfied by cache-to-cache transfers and different working set sizes; combining workloads gives insight into the different stresses placed on an architecture.

TABLE II
Workload Statistics

| | Percent of accesses resulting in a cache-to-cache transfer | | | # of 64 Byte blocks accessed |
|---|---|---|---|---|
| | all | clean | dirty | |
| TPC-W | 15% | 84% | 16% | 1,125 K |
| SPECjbb | 52% | 94% | 6% | 606 K |
| TPC-H | 69% | 43% | 57% | 172 K |
| SPECweb | 37% | 93% | 7% | 986 K |

## III. Cache Designs

The following section studies a variety of last level cache sharing arrangements to illuminate some of the pressures felt by the cache hierarchy.

### A. Private Last-Level Caches

With several cores, there are several design points for the last level cache configuration that we explore. On one end of the design spectrum are private caches. Here, each core is allowed exclusive access to an equally sized partition of the last level cache and shared data located on-chip can be accessed without going off-chip. Access latency and interconnect contention is low, but so is cache utilization; unused portions of the private caches are unavailable to the needs of other cores. Furthermore, since shared read-data is replicated across the private caches, the effective capacity is also lower [11], which has implications on the pressure put on the memory controllers.

### B. Shared Last-Level Cache

On the other end of the spectrum, all cores may share the last level cache in what we call a fully-shared cache [3, 23]. Access latency and interconnect contention are higher than in private caches, but since all of the cores can share all of the cache and there is no replication of data, the cache real-estate is efficiently utilized. However, there are serious concerns regarding the viability of a shared cache design. With a vanilla-LRU block replacement policy, there are no guarantees on any core's allocation in the cache and the scenario may arise where one core may significantly reduce the amount of cache memory available to another core [22]. Furthermore, the design of a large monolithic cache that satisfies many requests from many cores is a daunting challenge when engineers are faced with increasing wire delays [5].

### C. Shared-N-Way Last Level Caches

In the continuum, between private and shared, are shared-N-way last level caches. $N$ cores are statically assigned to share an equally-sized partition of the last level cache. The design strikes a balance between private and shared caches, by reaping the benefits of a shared structure while localizing its negative effects to a single partition. Utilization is not as high as the full-shared cache nor as low as the private caches and replication, though present, is not as prevalent as found in private-caches.

### D. Shared-N-Way Cache Scheduling Policies

The virtual machine hypervisor in a server consolidation system must allocate the virtual processors of each virtual image to the physical processors available in hardware. Whenever cores share caches, the scheduling policy assigning threads to cores also determines the assignment of threads to the shared-N-way caches. Co-scheduled threads reap the benefits of streamlined communication and efficient sharing of read-only blocks in the shared-N-way cache but may also experience thrashing if the combined working set exceeds available capacity. In this work, we explore four scheduling policies: round robin, affinity, a round-robin-affinity hybrid, and random.

- In round robin scheduling, load balancing is emphasized and each workload thread is assigned to a separate shared-N-way cache in a round robin fashion. The policy is best suited for spreading work across the chip and maximizing the threads' cache capacity when there are idle thread-contexts available.
- In affinity scheduling, sharing is maximized by attempting to schedule all workload threads on as few shared-N-way caches as possible. Affinity scheduling minimizes unnecessary data replication, as threads from the workload share data within a single (or a few) shared-N-way caches and provide faster access to shared data.
- In the round-robin-affinity scheduling hybrid, threads are scheduled in a round robin fashion with at least two threads from the workload sharing a shared-N-way cache.
- Random scheduling strives to capture the assignment of threads to shared-N-way caches that might be seen in an over-committed virtual machine. In this scenario, when threads are swapped in and out, they may be assigned to any available shared-N-way cache. After some period of time, the assignment may appear seemingly random.

Customarily, an operating system assigns a thread to a core. In consolidated servers running virtualization the assignments are more complex, the virtual machine manager

TABLE I
Workload Descriptions

| Workload | Description | Setup | Execution |
|---|---|---|---|
| SPECjbb | Order processing application for wholesaler Emphasizes the middle-tier business logic and performance of Java-based middleware | 3-tier client-server w/ six warehouses | 6400 requests w/ 15 seconds of warm-up time |
| SPECweb | World-wide web server | 3 tiers w/ Zeus Web Server 3.3.7 | 300 HTTP requests |
| TPC-H | Decision support | IBM DB2 v6.1 | Query #12 (shipping modes & order priority) on 512 megabyte database w/ 1 GB of memory |
| TPC-W | Web commerce modeling online bookstore | IBM DB2 v6.1 | Browsing mix for 25 web transactions |

or hypervisor must take the guest OS's thread-to-core assignments and find appropriate reassignments on the real hardware. Sun Microsystem's support for virtualization in the Solaris operating system [14] allows administrators to explicitly schedule threads to cores by statically binding the threads to specific cores, dynamically binding the threads to any core in a specific set, or by using an assignment heuristic of load-balancing, affinity, or utilization. The variety of scheduling algorithms we outline in our study try to replicate, as closely as possible, what an administrator might specify.

## IV. Methodology

### A. Simulation Environment

The experimental data presented in this paper was collected with PHARMsim, a full-system, cycle-accurate simulator built upon SIMOS-PPC [9,25]. The workloads simulated are isolated from one another through virtual machines; each workload runs a private copy of the AIX 4.3.1 operating system. Each workload is statically assigned its own portion of physical memory and has a completely private address space; no data is shared across workloads.

To efficiently simulate a variety of workload combinations, we use four core workload checkpoints that have been created and tuned for four cores. Workload checkpoints are snapshots of a workload taken after the simulator has booted and the workload has been installed and warmed; these checkpoints then run both user and operating system code for a specified number of transactions. Creating checkpoints alleviates the overhead of booting the operating system and ensures that the same set of transactions are run in each simulation.

Each four core workload checkpoint is loaded into the simulator at startup. Based on various scheduling algorithms, each thread from each workload is bound to a physical core at this time and remains bound throughout the simulation. The checkpoint may spawn additional threads; however the spawned threads are only permitted to run on the pre-allocated cores. We considered creating consolidated workload checkpoints for the variety of simulations we examined, but concluded the work involved would be prohibitive.

Our methodology is designed to mimic a dynamically partitioned system running a hypervisor or virtual machine. Additional hypervisor support allowing workloads to be dynamically remapped to different physical cores is left for future work. Currently, threads from a single workload can only be context switched within their domain of statically-assigned cores and not to other cores in the CMP. This methodology gives us greater control over thread placement and the ability to bind threads of a given workload to a certain set of physical cores.

The machine configuration used is found in Table III. With the exception of cache configuration, our machine model remains unchanged for our various simulations. In-order cores mimic Sun Microsystem's Niagara [23] and are indicative of future many-core architectural trends; out-of-order cores are left to future studies. The last level cache (L2) is set to a constant 16 megabytes in size and divided up accordingly:

- With private L2 caches, each core is assigned a 1 megabyte share of the aggregate 16 megabytes.
- With shared-N-way L2s, 2, 4, or 8 cores respectively share 2, 4, or 8 megabytes.
- With full-shared L2s, all 16 cores share the 16 megabytes.

We simulate a SGI Origin style directory protocol [24] with directory entries striped across the 16 cores by physical address to maintain coherence among the private caches. Each core is augmented with a directory cache to reduce the number of off-chip references. The interconnection network, which is also shared among workloads is a 2-D packet-switched mesh, with virtual channel flow control, dimension order routing and a 3-stage router pipeline. The router performs speculative virtual channel and switch allocation.

TABLE III
Machine Configuration

| | |
|---|---|
| Cores | 16 in-order |
| Interconnect | 2-D Packet-Switched Mesh |
| L0s (private) size/latency | 8KB/1 cycle |
| L1s (private) size/latency | 64 KB/2 cycles |
| L2s size/latency (shared by varying # of cores) | 16 MB/6 cycles |
| Memory latency | 150 cycles |
| Thread to core assignment | RR, Affinity, RR-Affinity, Random |

### B. Workload Combinations

Initially, we dedicated four cores and a fully shared 16MB cache to a single workloads and ran that workload in isolation to give a basis for comparison. Then, we ran several combinations of workloads and scheduling policies to study exactly how the behavior of workloads changed in the presence of other workloads. In all cases, the machine was filled to capacity, but never over-committed. If a workload happened to end prematurely, it was restarted to keep the system at capacity, allowing us to study the system in a steady state. The combination of workloads and policies are tabulated in Table IV with the number of replicated instances given in parenthesis. Due to issues with the workload driver, SPECweb could not be combined in the heterogeneous mixes.

TABLE IV
EXPERIMENTAL RUNS

| Heterogeneous Mixes | |
|---|---|
| Mix 1 | TPC-W (3) & TPC-H (1) |
| Mix 2 | TPC-W (2) & TPC-H (2) |
| Mix 3 | TPC-W (1) & TPC-H (3) |
| Mix 4 | SPECjbb (3) & TPC-H (1) |
| Mix 5 | SPECjbb (2) & TPC-H (2) |
| Mix 6 | SPECjbb (1) & TPC-H (3) |
| Mix 7 | SPECjbb (3) & TPC-W (1) |
| Mix 8 | SPECjbb (2) & TPC-W (2) |
| Mix 9 | SPECjbb (1) & TPC-W (3) |
| Homogeneous Mixes | |
| Mix A | TPC-W (4) |
| Mix B | TPC-H (4) |
| Mix C | SPECjbb (4) |
| Mix D | SPECweb (4) |

We focus our simulations on a 16-core system to make simulation time tractable given the large number of workload combinations. However, scaling to larger systems and observing the effect on the trends we present is an avenue for future work.

## V. Results

To provide insight into the behavior of server consolidation workloads we present three metrics: single-workload performance, miss rate, and miss latency. Workload instances in the consolidated mixes are normalized to a single workload instance run in isolation with four cores and 16 MB of fully shared last level cache. Since cycle-per-instruction is not a meaningful performance measurement in non-deterministic multi-threaded execution, we choose to use runtime values (normalized cycle count) to give the reader a sense of a single workload's performance. Analyzing the performance of the entire system running multiple server consolidation workloads is problematic as each workload executes a different number of transactions and transactions across each benchmark are of different sizes. Therefore we limit our performance evaluation to looking at the relative change in performance of each virtual machine in the different workload mixes; this metric is similar to the cycles-per-transaction used by [28]. For our performance evaluation we use statistical simulation methods as described by Alameldeen and Wood [1]. Miss rates presented are for the last level cache misses seen by each virtual machine (as opposed to the miss rate seen by the individual last level caches). Presenting the miss rate observed by each cache would obscure the impact consolidation has on each individual workload. Miss latency is defined as the time to satisfy a miss to the last level of private cache (L1). This measurement includes the latency incurred by L1 cache-to-cache transfers, L2 accesses, and the latency of a memory access. The commercial workloads studied are sensitive to miss latency; showing the relative latency adds to the overall performance picture.

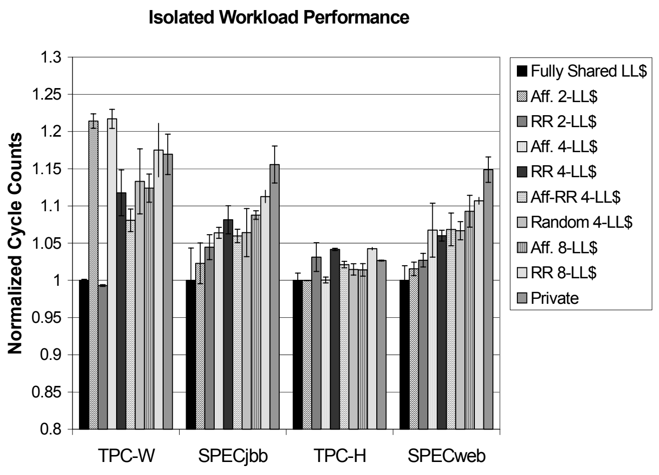

Fig. 2. Performance of workloads run in isolation

### A. Single Isolated Workloads

Observing each workload in isolation helps with understanding the effect of consolidation on the workload. We simulate a single instance of each workload with varying degrees of cache sharing and a variety of scheduling policies running. In each simulation, the entire system is dedicated to the workload; four cores (of the 16) are active while the remaining 12 remain idle. Affinity usually helps with sharing, though effective capacity is reduced; Round Robin allows the whole chip's cache to be utilized by threads, while affinity limits the shared-8-way and shared-4-way case to 1/2 and 1/4 of the cache respectively. In most cases, as expected, Figure 2 shows that single workloads performance degrades as the on-chip cache storage is decreased (i.e. as *N* decreases in the shared-N-way expression). In Figure 2 2-LL$ refers to a setup with 2 last level caches or a shared-8-way system. Round robin scheduling distributes the threads around the chip; this placement achieves a more even distribution of traffic on the interconnection network. Affinity scheduling yields significant contention for TPC-W as it has a large data footprint; alleviating pressure on the interconnect improves performance. Interconnect latency is 20% lower for round robin scheduling than for affinity scheduling.

Figure 3 shows a corresponding increase in misses as the last level cache capacity seen by each thread decreases. In the case of the shared-4-way caches, the round robin scheduling policy (which maximizes on-chip cache capacity) has the worst miss rate as it is forced to replicate read-shared data. Affinity scheduling maximizes the positive effects of sharing; however, the miss rate is slightly higher

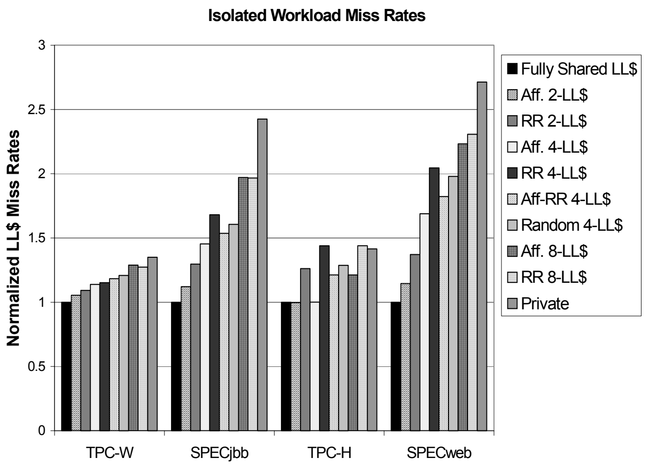

Fig. 3. Miss rates for workloads run in isolation

than affinity in the shared-2-way configuration for TPC-W and SPECjbb. The cache capacity in this setup is very constrained causing an increase in conflict and capacity misses among the threads.

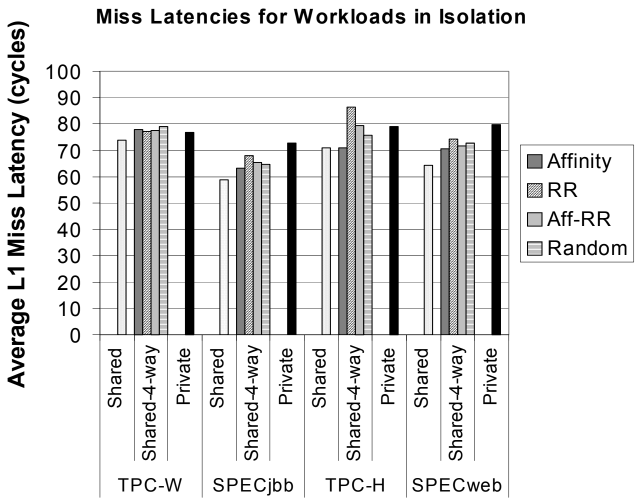

Fig. 4. Miss latencies of workloads run in isolation

Figure 4 shows the average miss latencies for each workload run in isolation for each scheduling and 3 different cache configurations: shared, shared-4-way and private.

## *B. Effects of Scheduling on Homogeneous Workload Mixes*

In Figures 5, 6, and 7 we compare the effects of different scheduling algorithms in terms of performance, miss latency and miss rates of the homogeneous mixes. Affinity scheduling is the best policy since it can avoid additional long latency misses by sharing data in the same last level cache. TPC-W performs best from a random placement of threads; this placement reduces interconnect congestion. SPECjbb and SPECweb show significant performance degradation (Figure 5) for round robin scheduling. However, in a long-running, real system, scheduling patterns similar to affinity-round-robin (aff-rr) or random are likely to emerge. The miss latencies are normalized to miss latency of each workload running in isolation with affinity

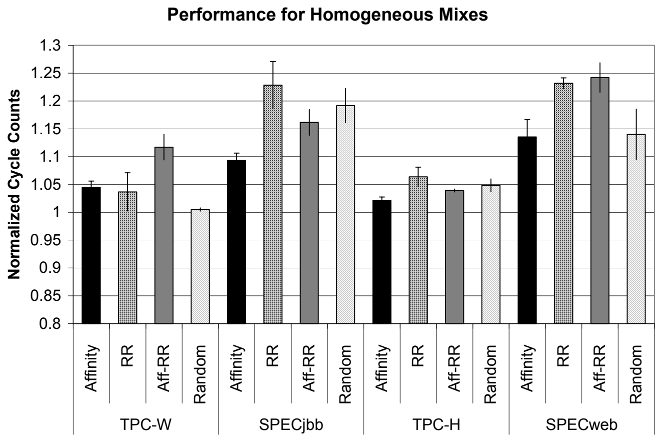

Fig. 5. Single-Workload Performance for Homogeneous Mixes Relative Workload in Isolation

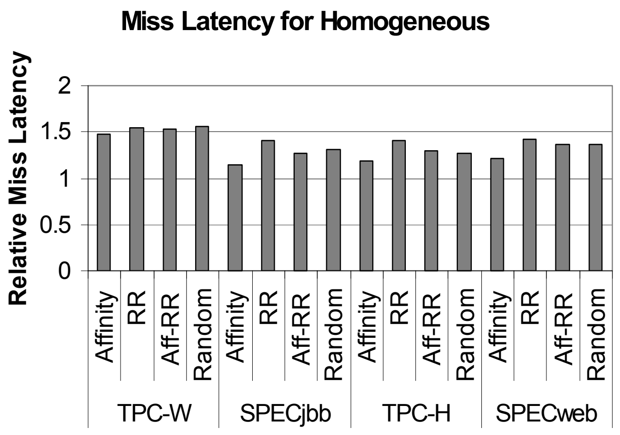

Fig. 6. Effect of Thread Scheduling Policies on Miss latency

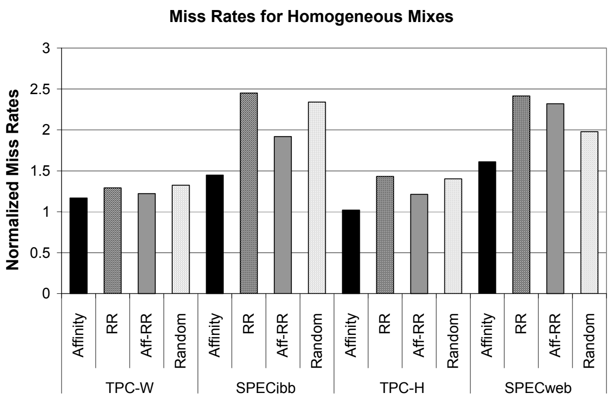

Fig. 7. Miss rates of Homogeneous Mixes relative to workloads run in isolation

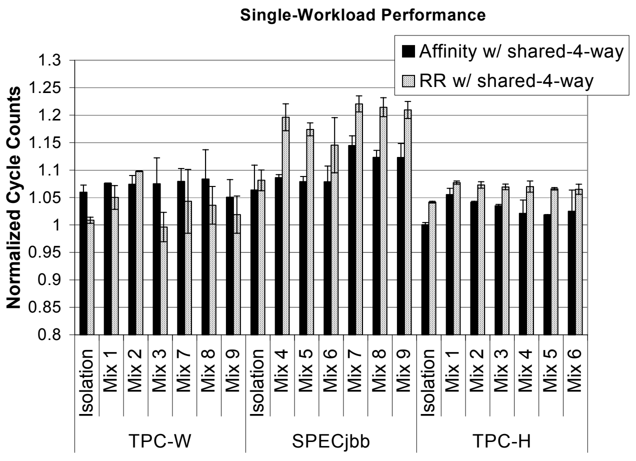

Fig. 8. Single-Workload Performance of Heterogeneous Mixes relative to workloads run in isolation

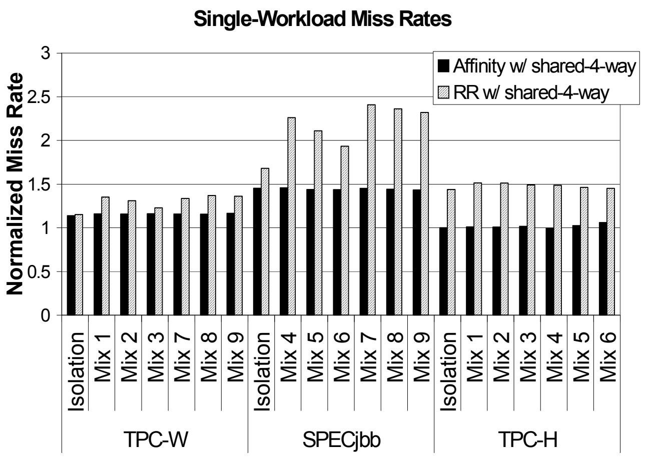

Fig. 9. Single-Workload Miss Rates of Heterogeneous Mixes relative to workloads run in isolation

scheduling. From these simulations, we observe that if the workload has the whole chip (isolation), affinity scheduling does slightly better than round-robin. In the round-robin case, dirty misses need to travel further to be satisfied, while in the affinity case, the communicating cores are grouped closely together allowing a faster response for a request to a dirty block. Going from isolated runs to homogeneously mixed runs, TPC-W shows the greatest increase in miss latency as it has a large memory footprint and causes significant thrashing in the cache when it is forced to compete for cache space. Figure 7 shows an increase in miss rate for the workloads when they compete for resources which is expected and accounts for increases in miss latency. Competion for cache resources due to conflicts spills over into the interconnect and the memory controllers which can adversely affect performance through greater interference.

## C. Heterogeneous Workload Mixes

In Figure 8, we show the cycle counts for a single workload instance in each consolidated mix normalized to the run time in isolation with a fully shared 16MB cache. The labels on the x-axis correspond to the workload mixes given

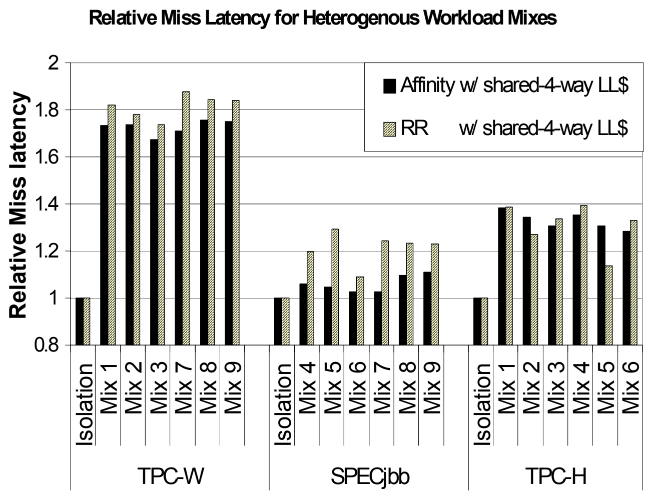

Fig. 10. Miss latencies of Heterogeneous Mixes relative to workloads run in isolation

in Table IV. The performance for affinity shared-4-way and round robin shared-4-way in isolation is shown to illustrate to what extent performance of one workload is isolated from another. The performance of TPC-H is largely unaffected by the presence of other workloads on the chip; the relatively smaller footprint coupled with the larger percentage of accesses that are satisfied by private cache-to-cache transfers make TPC-H less susceptible to interference by other workloads. SPECjbb, on the other hand, sees large performance degradation in the presence of other workloads and shows high variability depending on how workloads are combined. SPECjbb's performance degradation can be attributed to the large increase in miss rate shown in Figure 9; conversely, TPC-H with affinity scheduling sees almost no increase in miss rate with respect to a 16 MB cache.

Figure 10 shows the average miss latencies for the different workload mixes. The latencies are separated by workloads in the mix and normalized to the workload's latency in isolation with affinity scheduling and a shared-4-way cache. We expect to see higher relative miss latencies when we run in a consolidated environment, but not all workloads are affected the same. SPECjbb's miss latency is the least sensitive to the other workloads running concurrently, while TPC-W is the most sensitive. Miss latency in SPECjbb may be less affected than other workloads, but combining the small increase in miss latency with the large increase in miss rate results in performance degradation. The wide range of miss latencies demonstrated by the data indicate that the workloads are incredibly sensitive to the co-scheduled workloads.

## D. Effects of Cache Configuration on Average Miss Latency

We vary the number of cores that share a last level cache; the changes in miss latency for different degrees of sharing are shown in Figure 11 for the heterogeneous mixes. The results shown here are limited to affinity-based scheduling and normalized to the shared-4-way isolation latencies. TPC-H has the lowest average miss latency when it is configured as shared-4-way last level caches, each cache with its own workload. This configuration has no replication and provides low latency for the read-shared data and also prevents interference from other workloads with larger data footprints. The shared-8-way cache configuration (2 8MB caches) allows more flexible use of cache space which provides better performance for SPECjbb in different workload mixes especially when combined with TPC-H which puts less pressure on the caches. With only 2 caches, TPC-H has to share space with other workloads and suffers as a result. For TPC-W and SPECjbb, fewer caches minimize replication and provides a more effective use of cache space.

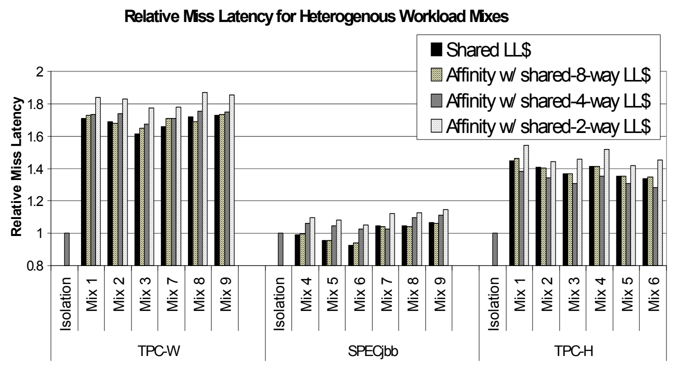

Fig. 11. Varying the Degree of Sharing for Partially Shared Caches for Heterogeneous Mixes

## E. *Replication and Cache Utilization*

Figure 12 shows the percentage of total cache lines that are replicated across the last level caches. The rightmost bar shows the maximum replication which occurs when each thread has its only private last level cache. Affinity scheduling is left off this graph as it will have no replication in the shared-4-way configuration. Round robin has the most replication as each thread of the workload is in a different last level cache. Affinity-round robin and random have less replication and therefore make more effective use of the overall last level cache capacity. SPECjbb and SPECweb have the most replication in the round-robin scheduling simulation. 73% and 64% of the last level cache lines, respectively, are not replicated in any other cache. Replicating these read-shared lines, particularly in SPECjbb and SPECweb, can significantly reduce the effective cache capacity and consequently hurt performance. SPECweb sees more replication in the isolated case since there is less competition for cache capacity among threads.

Figure 13 shows the percentage of total last level cache capacity that each workload occupies in each of the heterogenous mixes. In order to exacerbate the effects of collocation, we used the round-robin scheduling policy to collect the data; this data is sampled after 500 million instructions have run on the CMP. The configuration shown here is four caches that are shared by four threads each (shared-4-way). TPC-H workloads occupy less than their fair share of capacity (25%) in almost all caches which can account for higher miss latencies as seen in Figure 10. However, TPC-H places smaller demands on the cache. Given an overall increase in miss latency, TPC-H's overall cache demands are not being

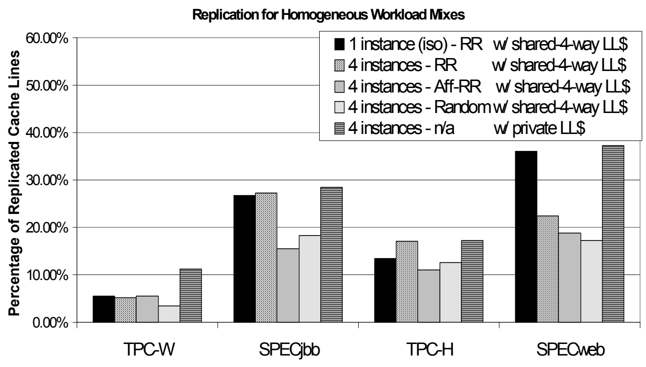

Fig. 12. Percentage of replicated lines in last level cache for Homogeneous Mixes (Snapshot at 500 million instructions)

met to deliver the same performance it achieves with round robin scheduling in isolation. Similarly, SPECjbb always shares capacity equally with other copies of the same workload. Mix 7, which runs three copies of SPECjbb and one copy of TPC-W, allocates only a small portion of the last level caches to the first copy of SPECjbb. SPECjbb sees a large increase in its miss rate (Figure 9) when combined with TPC-W in Mixes 7 through 9 due to the significant pressure both workloads place on the cache resulting in a large number of conflicts.

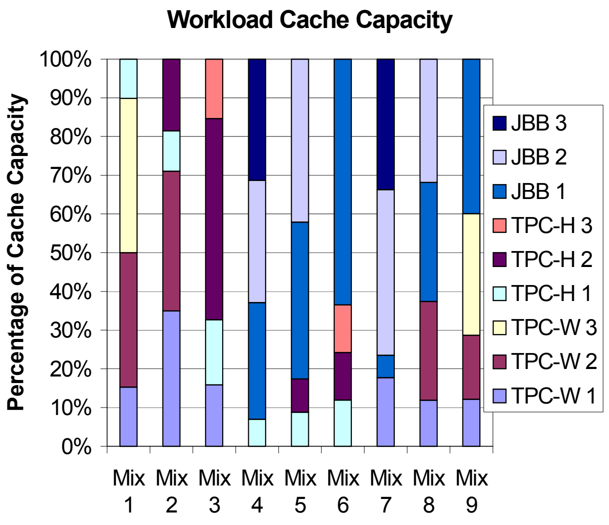

Fig. 13. Snapshot of Cache Utilization per Workload for Heterogeneous Mixes

## VI. Related Work

Since its unveiling, Simultaneous Multi-threaded (SMT) processor research has concerned itself with how to best-manage shared resources and control interactions between threads. In the beginning, the focus was on assigning threads to pipeline resources within the core [36,37]. Later, the work extended into SMT's shared memory hierarchy when Lo showed that database applications, which have inter-thread sharing, can benefit from SMT [27].

Since threads on multi-core chips only interact when memory requests leave the core and enter the memory hierarchy, the interactions between threads are not as pronounced as in SMT. Nonetheless, research has begun to emerge. Cooperative Caching, Victim Replication, and Adaptive Selective Replication [4, 10, 38] are caching techniques that recognize threads interactions and ultimately strive to achieve high throughput while hiding wire delay. Similarly, papers written by Liu, Suh, and Blelloch [8, 11, 26, 33], all try to optimally manage a multi-core's cache for performance gains.

Taking a stance of quality of service over throughput, Nesbit et. al [29] described the extent to which a single-threaded application can monopolize the memory controller and cause harm to a co-scheduled single-threaded applications. Other proposals by Iyer, Nesbit, Guo, and Kim [15, 19, 20, 22, 30] are in the same vein and describe mechanisms guaranteeing an architectural quality of service in multi-core chips.

The work in this paper is similar to the previous work in that it highlights the importance of thread interaction. Unlike previous work, which characterized interactions in single commercial workloads or multiprogrammed scientific workloads, this work characterizes interactions in server consolidation workloads.

Recent work by Marty and Hill [28], highlights one unique research avenue that has emerged from studying server consolidation workloads. This work explores coherence protocol optimizations and is one of the first to look at the architectural opportunities brought to light by server consolidation workloads. Server consolidation workloads will continue to facilitate new research directions.

Like the work presented here, Hsu's large scale CMP work [16] also looks at multi-core chips running server consolidation workloads. Hsu's focus is finding the best cache configuration for large scale CMPs, given a commercial workload. Conversely, our work is a characterization of how individual workloads behave in isolation and in the company of other workloads, given a variety of cache configurations. Furthermore, the numbers presented in this study come from a sophisticated full-system execution driven simulator, while Hsu's numbers are a worst-case bound obtained from a trace-driven simulation.

## VII. Future Work

Looking forward, consolidated servers are likely to have many-cores on a chip. While our work gives an indication as to how workloads perform in a consolidated environment, it is only on the scale of 16-core chips. Studying higher degrees of consolidation, either by increasing the number of threads in a workload or increasing the number of workloads running, would allow researchers to accurately forecast behavior even further into the future. Additionally, we study workloads with the same number of threads (but different working set sizes); consolidating workloads with different numbers of threads is also worth evaluating.

The behavior expressed by the workloads combinations presented in this paper may be dependent upon how the specific phases of workloads interacted with each other. It is possible that by doing some phase analysis and aligning different combinations of phases from different workloads that one can study the interactions in more depth. Such an analysis would give, for example, an indication of the range of interference.

Finally, the work presented here performs scheduling by statically binding threads to cores at startup. Since our results indicate the scheduling of threads to shared resources is important, we would like to study the effects of schedulers dynamically adjusting assignments, in response to context-switches and changing demands of the system. A more sophisticated hypervisor implementation will also give us the opportunity to study how an over-committed system behaves.

## VIII. Conclusions

Studying the behaviors of a multi-commercial workload environment is interesting and relevant to the industrial and research communities. As core-rich chips become more prevalent, our work indicates that the status quo approaches to managing shared resources, especially caches, has profound effects on the individual and overall behavior of the workloads. Our results provide indicators that suggest both performance and fairness are affected, and that the impact of the approach to scheduling threads on shared-resources is especially important.

A large body of work exists studying the interactions that go on across single-threaded workloads and within a single multi-threaded workload. This work distinguishes itself from prior work by observing the interaction of multiple multi-threaded workloads (server consolidation) in an execution-driven simulator. The results presented here reinforce intuition about constructive/destructive interaction, but most notably, the interaction is quite dramatic. We quantify this behavior and believe our work forms a motivating basis for emerging and up-and-coming research. Other methodological considerations, such as workload start times deserve further exploration.

This work stresses the importance of studying server consolidation workloads in the designing of multi-core architectures. We put forth a methodology of running multiple workloads in a virtual machine setup to isolate the workloads. Our current methodology does not fully explore the workload space; temporal effects of combining workloads is an area for future research. Our results indicate that most workloads benefit from the reuse of shared data in the last level cache, as evidenced by the affinity scheduling policy. One of the main objectives of this study is to further the communities understanding of cache interactions among different commercial workloads; to this end, we have presented cycle counts, miss rates, and miss latencies for a variety of different workload combinations and the impact on single-workload performance that the various combinations can have. When workloads with different cache and memory requirements are combined fairness issues need to be considered.

In a consolidated server, virtual machines are guaranteed functional isolation from one another. Our results, showing that the behavior of one virtual machine may affect the other, suggests that perhaps a guarantee of apparent workload isolation in a consolidated should feasibly extend from functional isolation into performance isolation.

## IX. Acknowledgments

We would like to thank our anonymous reviewers for their helpful feedback and in particular Ravi Iyer for his suggestions on this work. Various members of the Pharm group have also provided useful comments. This research was supported in part by the National Science Foundation under grants CCR-0133437, CCF-0429854, and CCF-0702272, an IBM PhD Fellowship and a NSF Graduate Fellowship, as well as grants and equipment donations from IBM and Intel.

## References

[1] A. R. Alameldeen and D. A. Wood, "Variability in architectural simulations of multi-threaded workloads," in *Proceedings of the 9th Annual International Symposium on High Performance Computer Architecture*, 2003.

[2] L. A. Barroso, K. Gharachorloo, and E. Bugnion, "Memory system characterization of commercial workloads," in *ISCA '98: Proceedings of the 25th annual international symposium on Computer architecture*, 1998, pp. 3–14.

[3] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: a scalable architecture based on single-chip multiprocessing," in *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, 2000, pp. 282–293.

[4] B. M. Beckmann, M. R. Marty, and D. A. Wood, "ASR: Adaptive selective replication for CMP caches," in *MICRO '06: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 443–454.

[5] B. M. Beckmann and D. A. Wood, "Managing wire delay in large chip-multiprocessor caches," in *Proceedings of the 37th International Symposium on Microarchitecture*, 2004.

[6] T. Bezenek, T. Cain, R. Dickson, T. Heil, M. Martin, C. McCurdy, R. Rajwar, E. Weglarz, C. Zilles, and M. Lipasti, "Characterizing a Java implementation of TPC-W," Workshop presentation at CAECW'00 held in conjunction with HPCA, 2000.

[7] M. V. Biesbrouck, B. Calder, and L. Eeckhout, "Considering all starting points for simultaneous multithreading simulation," in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2006.

[8] G. E. Blelloch and P. B. Gibbons, "Effectively sharing a cache among threads," in *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, 2004, pp. 235–244.

[9] H. Cain, K. Lepak, B. Schwarz, and M. H. Lipasti, "Precise and accurate processor simulation," in *Workshop On Computer Architecture Evaluation using Commercial Workloads*, February 2002.

[10] J. Chang and G. S. Sohi, "Cooperative caching for chip multiprocessors," in *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*, 2006, pp. 264–276.

[11] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Optimizing replication, communication and capacity allocation in CMPs," in *Proceedings of the 32nd International Symposium on Computer Architecture*, June 2005.

[12] M. Dubois, J. Jeong, and A. Nanda, "Shared cache architectures for decision support systems," *Perform. Eval.*, vol. 49, no. 1-4, pp. 283–298, 2002.

[13] R. Figueiredo, P. A. Dinda, and J. Fortes, "Guest editors' introduction: Resource virtualization renaissance," *Computer*, vol. 38, no. 5, pp. 28–31, 2005.

[14] H. J. Foxwell, M. Lageman, J. P. van Hoogeveen, I. Rozenfeld, S. Setty, and J. Victor, "The Sun blueprints guide to Solaris containers: Virtualization in the Solaris operating system," Miscellaneous Sun Microsystems Technical Report, Sun Microsystem, Tech. Rep., October 2006.

[15] F. Guo, H. Kannan, L. Zhao, R. Illikkal, R. Iyer, D. Newell, Y. Solihin, and C. Kozyrakis, "From chaos to QoS: case studies in CMP resource management," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 21–30, March 2007.

[16] L. Hsu, R. Iyer, S. Makineni, S. Reinhardt, and D. Newell, "Exploring the cache design space for large scale CMPs," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 24–33, 2005.

[17] J. Huh, J. Chang, D. Burger, and G. S. Sohi, "Coherence decoupling: Making use of incoherence," in *Proceedings of ASPLOS 2004*, 2004.

[18] Intel, "From a few cores to many: A Tera-scale computing research overview," 2006. [Online]. Available: http://download.intel.com/research/platform/terascale/ terascale_overview_paper.pdf

[19] R. Iyer, L. Zhao, F. Guo, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. Hsu, and S. Reinhardt, "QoS policies and architecture for cache/memory in CMP platforms," in *Proceedings of the 2007 ACM SIGMETRICS international Conference on Measurement and Modeling of Computer Systems*, 2007.

[20] R. Iyer, "CQoS: a framework for enabling qos in shared caches of CMP platforms," in *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, 2004, pp. 257–266.

[21] J. Jann, L. M. Browning, and R. S. Burugula, "Dynamic reconfiguration: Basic building blocks for autonomic computing on IBM pSeries servers," *IBM Syst. J.*, vol. 42, no. 1, pp. 29–37, 2003.

[22] S. Kim, D. Chandra, and Y. Solihin, "Fair cache sharing and partitioning in a chip multiprocessor architecture," in *PACT '04: Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, 2004, pp. 111–122.

[23] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way multithreaded Sparc processor," *IEEE Micro*, vol. 25, no. 2, pp. 21–29, 2005.

[24] J. Laudon and D. Lenoski, "The SGI Origin: a ccNUMA highly scalable server," in *Proceedings of the 24th International Symposium on Computer Architecture*, 1997, pp. 241–251.

[25] K. M. Lepak, H. W. Cain, and M. H. Lipasti, "Redeeming ipc as a performance metric for multithreaded programs," in *Proceeding of 12th International Conference on Parallel Architectures and Compilation Techniques*, 2003, pp. 232–243.

[26] C. Liu, A. Sivasubramaniam, and M. Kandemir, "Organizing the last line of defense before hitting the memory wall for CMPs," in *HPCA '04: Proceedings of the 10th International Symposium on High Performance Computer Architecture*, 2004, p. 176.

[27] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy, and S. S. Parekh, "An analysis of database workload performance on simultaneous multithreaded processors," in *ISCA '98: Proceedings of the 25th annual international symposium on Computer architecture*, 1998, pp. 39–50.

[28] M. Marty and M. Hill, "Virtual hierarchies to support server consolidation," in *Proceedings of the 34th International Symposium on Computer Architecture*, 2007.

[29] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith, "Fair queuing memory systems," in *MICRO '06: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 208–222.

[30] K. J. Nesbit, J. Laudon, and J. E. Smith, "Providing QoS with virtual private machines," in *Proceedings of the 1st Workshop on Chip Multiprocessor Memory Systems and Interconnects*, Feb 2007.

[31] P. Ranganathan, K. Gharachorloo, S. V. Adve, and L. A. Barroso, "Performance of database workloads on shared-memory systems with out-of-order processors," in *Architectural Support for Programming Languages and Operating Systems*, 1998, pp. 307–318. [Online]. Available: citeseer.ist.psu.edu/ranganathan98performance.html

[32] J. E. Smith and R. Nair, "The architecture of virtual machines," *Computer*, vol. 38, no. 5, pp. 32–38, 2005.

[33] G. E. Suh, L. Rudolph, and S. Devadas, "Dynamic partitioning of shared cache memory," *J. Supercomput.*, vol. 28, no. 1, pp. 7–26, 2004.

[34] Systems Performance Evaluation Cooperative, "SPEC benchmarks," http://www.spec.org.

[35] Transaction Processing Performance Council, "TPC benchmarks," http://www.tpc.org.

[36] D. M. Tullsen and J. A. Brown, "Handling long-latency loads in a simultaneous multithreading processor," in *MICRO 34: Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, 2001, pp. 318–327.

[37] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: maximizing on-chip parallelism," in *ISCA '98: 25 years of the international symposia on Computer architecture (selected papers)*, 1998, pp. 533–544.

[38] M. Zhang and K. Asanovic, "Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors," in *Proceedings of the 32nd International Symposium on Computer Architecture*, 2005, pp. 336–345.