Appears in the 19th International Conference on Supercomputing

A NUCA Substrate for Flexible CMP Cache Sharing

Jaehyuk Huh Changkyu Kim[†] Hazim Shafi Lixin Zhang [§]
Doug Burger Stephen W. Keckler[†]

[†]Dept. of Computer Sciences
The University of Texas at Austin

[§]Austin Research Laboratory
IBM Research

ABSTRACT

We propose an organization for the on-chip memory system of a chip multiprocessor, in which 16 processors share a 16MB pool of 256 L2 cache banks. The L2 cache is organized as a non-uniform cache architecture (NUCA) array with a switched network embedded in it for high performance. We show that this organization can support the spectrum of degrees of sharing: *unshared*, in which each processor has a private portion of the cache, thus reducing hit latency, *completely shared*, in which every processor shares the entire cache, thus minimizing misses, and every point in between. We find the optimal degree of sharing for a number of cache bank mapping policies, and also evaluate a per-application cache partitioning strategy. We conclude that a static NUCA organization with sharing degrees of two or four work best across a suite of commercial and scientific parallel workloads. We also demonstrate that migratory, dynamic NUCA approaches improve performance significantly for a subset of the workloads at the cost of increased power consumption and complexity, especially as per-application cache partitioning strategies are applied.

Categories and Subject Descriptors: C.1.2 [Processor Architectures]: Multiple Data Stream Architectures (Multiprocessors), B.3.2 [Memory Structures]: Design Styles – Shared memory, C.4 [Performance of Systems] – Design studies

General Terms: Performance, Design, Experimentation, Measurement

Keywords: chip-multiprocessor, cache sharing, non-uniform cache architecture

1. INTRODUCTION

Assuming that power ceilings do not limit continued CMP integration, future server-class processors are likely to contain large numbers of processors along with large caches. While level-one caches are likely to remain private to each processor, the degree of sharing for the integrated level-two caches will be a key design decision. The level-two caches may be shared by all processors, or may be separated into private per-processor partitions, or any point in between. In this paper, we explore this range of caching and sharing policies for a CMP targeted at 45 nanometer technologies, with 16MB of on-chip cache and 16 high-capability processors. The specific layout we propose, based on a non-uniform cache architecture (NUCA) designs, mitigates the effects of growing wire resistivity and thus intra-cache communication delays, and permits any degree of cache sharing in this single implementation. The underlying hardware is sufficiently flexible to have its degree of sharing chosen dynamically, adjusted by the operating system.

The tension between a greater versus lesser degree of sharing is driven by the reduced misses of greater cache sharing versus the reduced hit latencies of lesser cache sharing. More precisely, we call the *sharing degree* the number of processors that share a given pool of cache. In this terminology, a sharing degree of one means that each processor has its own private L2 partition, whereas a sharing degree of sixteen means that all processors are sharing a single large cache array.

Greater sharing degrees reduce misses in two ways. First, they reduce the number of shared copies of a single line existing on-chip, since each line maps to only one place in shared caches. Second, they provide a larger shared pool to tolerate imbalances across the sharers' working sets. However, larger sharing degrees result in longer cache latencies, as the shared cache is larger than the individual private partitions, assuming total cache area is held constant. An ideal design would somehow capture the benefits of both reduced misses *and* reduced hit latencies.

As shown in Figure 1, the floorplan we evaluate consists of a single large cache array, broken into numerous banks which are connected by a lightweight, switched 2-D mesh network. Processors span the top and bottom edges of the cache array, and each has a port into the L2 cache network. By adjusting the bits used to route memory addresses to a cache bank, the cache array is configurable by the system to use any degree of sharing. If each processor maps the same address bit string to a different bank, the sharing degree is one. If all processors map the same address bits to a single bank, the sharing degree is sixteen.

We first measured the optimal bank size and sharing degree for a cache in which the mapping of lines to banks does not change (this uniprocessor organization was called static

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICS'05, June 20–22, 2005, Boston, MA, USA.
Copyright 2005 ACM 1-59593-167-8/06/2005 ...$5.00.

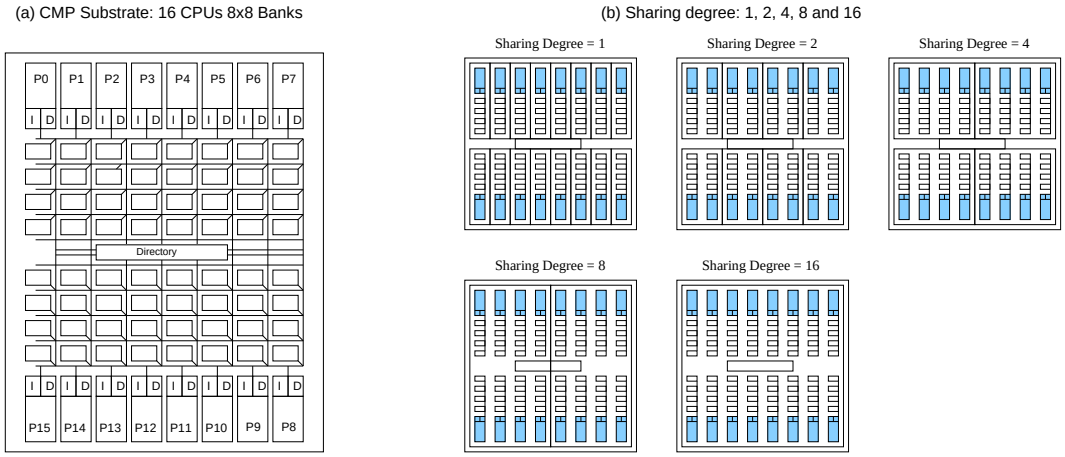

Figure 1: 16 Processor CMP Substrate

NUCA, or S-NUCA, in prior work [13]). We then evaluated per-application sharing degrees, as well as different techniques, such as intra-cache line migration (called dynamic, or D-NUCA caches in prior work) and level-one prefetching, to reduce average L2 hit latencies and thus support larger sharing degrees. The results of this study are the following:

- The ideal bank size across all configurations is 64KB.
- We confirmed that significant latency reductions are possible for private L2 caches, and significant miss reductions are possible for shared L2 caches. The results showed 54% latency reduction with private L2 caches and 33% reduction of external memory accesses with fully shared caches.
- For shared S-NUCA organizations, low-to-medium sharing degrees, from one to four, provide the best performance for all applications except one. The best sharing degree across all benchmarks is four.
- Although dynamic mapping (D-NUCA) reduces hit latencies for high sharing degrees by 30%, performance improvement is relatively modest except two applications with 17% and 20% speedups. However, this improvement incurs considerable complexity and power overheads.
- Level-one prefetching improves performance proportionately to the L2 cache hit delay, but does not change the ideal sharing degree significantly.
- A priori, application-specific adaptation of the sharing degree provides an average 5% performance boost for the D-NUCA organization, but little gain for the S-NUCA organization.
- Per-line sharing degree, which provides two different sharing degrees for private and shared cache blocks, can mitigate the negative effect of large sharing degrees, thus reducing overall miss rates and avoiding slow hit latencies. The best private sharing degree is one or two, reducing hit latencies for private blocks, but the best shared sharing degree is 16, maximizing caching efficiency.

In Section 2, we describe the flexible NUCA substrate that supports reconfigurable sharing degrees. In Section 3, we describe the simulation environment that we use, evaluate per-application and per-class sharing degrees, and compare the performance of S-NUCA and D-NUCA. We also discuss the effect of L1 prefetching on the best sharing degree. In Section 4, we describe related work in CMP L2 cache designs. We conclude in Section 5.

2. CMP L2 CACHE DESIGN SPACE

Figure 1 (a) shows a multiprocessor chip projected for 45nm process technology in 2010. The chip has 16 processors and 16MB of L2 cache capacity on a die area of approximately $300mm^2$. The L2 cache resides in the middle of the chip and contains an array of banks connected by a lightweight routing network. Half of the 16 CPUs are at the top and the other half are at the bottom. Because of the large number of CPUs and cache banks, there are many possible design options for the L2 cache. In this paper, we search for the cache sharing organization that gives the best average performance across a range of commercial and scientific applications.

In the rest of this section, we first discuss the implications of varying the number of CPUs that can share a cache bank. We then describe both static and dynamic ways to map lines to cache banks. We then propose two approaches for reducing L2 latency and facilitating larger shared cache pools (thus reducing both latency *and* misses): an efficient migration policy that moves data to tiles closer to the requesting CPU, and partial tag arrays to speed up the lookup of remote cache banks.

2.1 Sharing Degree

Shared L2 caches have been adopted as an alternative to traditional private L2 caches for CMP L2 cache designs [3, 7, 11, 23]. One of the key factors in CMP shared caches is the number of CPUs that can share a L2 cache, called the *sharing degree* (SD for short). A sharing degree of N means that N CPUs share a L2 cache.

The basic trade-offs of varying the sharing degree are hit latency, hit rate, inter-processor communication, and coherence maintenance overhead. In general, for hit latency, a lower sharing degree is better as each L2 cache is

smaller. For hit rates and inter-processor communication, higher sharing degrees are better because they make more efficient use of cache capacity. Since a higher sharing degree has more L1 caches sharing an L2 cache, it is more expensive to maintain L1 coherence. However, a lower sharing degree means more discrete L2 caches on a chip, making maintaining L2 coherence more expensive.

The main advantage of higher sharing degrees is higher L2 cache hit rates. If the working sets across CPUs are not well balanced, private L2 caches can make one CPU suffer from capacity misses while other CPUs have unused cache space. Shared caches, on the other hand, allow that otherwise unused cache space to be used by the space-hungry CPU. Furthermore, shared caches keep at most one copy of a block, not wasting space by storing multiple copies of the same block, unlike private L2 caches sharing copies of the same line. As a result, shared caches can effectively store more data, indirectly increasing hit rates. A shared cache also avoids the L2 coherence misses generated by private L2 caches [8].

Inter-processor communication through a shared L2 cache is normally faster than through private L2 caches connected by a coherent bus. With shared L2 caches, processors communicate through L2 cache blocks directly. With private L2 caches, processors have to communicate through private L2 caches and coherence fabric.

The main drawback of a higher sharing degree is the potential for higher average hit latency due to the larger size, longer wire delays, and increased bandwidth requirements. In future wire-dominated implementations, the effect of increased hit latency may outweigh the benefit of increased hit rate.

Another overhead of shared caches is that each L2 cache needs to maintain coherence for the L1 caches sharing the L2 cache. In this study, the system maintains L1 cache coherence by embedding sharing status vectors in the L2 tag arrays. The tag of an L2 cache line includes a bit mask to indicate which L1 caches have copies of the line. When a L2 cache receives an update request from an L1 cache, it sends invalidation messages to other L1 caches that have a copy of the requested block. Such directory-based L1 coherence was used in the Piranha CMP [3]. Although broadcast-based protocols can reduce the area overhead for sharing vectors [11, 23], we used the directory protocol because it is more scalable for these types of systems, demanding less L2-to-L1 bandwidth than broadcast protocols.

A lower sharing degree means that each chip will have more private L2 caches, thus it is more difficult to maintain L2 data coherence. As with the L1 caches, we use a centralized L2 tag directory to maintain on-chip L2 coherence. When a L2 miss is detected, the request is sent to the centralized L2 tag directory. The directory decides, without snooping other L2 caches in the chip, whether to get data from another L2 cache on the chip or whether to issue an off-chip memory request. The directory-based coherence protocol has a number of advantages over broadcast-based protocols. First, it relieves the coherence bus from becoming the bottleneck. Second, it detects on-chip cache misses faster because it does not need to snoop other L2 caches in the chip. Third, the directory functions as the single external coherence snoop point for requests from other chips, avoiding the need to have multiple L2 caches on the same chip snoop the global bus.

2.2 Organization

Shared caches require more bandwidth than private caches, since the data request rate is proportional to the number of processors. Two common ways of increasing bandwidth are i) adding more access ports and ii) splitting a monolithic cache into multiple independently operated banks. L2 caches typically use the latter because it is more cost-effective than having multiple ports. For instance, the dual-processor Power4 CMP [23] uses three banks, and the eight-processor Piranha CMP uses eight banks for each L2 cache.

As exemplified in Figure 1, a CMP chip can consist of many independently operated L2 cache banks. To enable high-speed clocking and reduce the space for wires, we use a switched network, instead of traditional dedicated wires, to connect cache banks and processors. A processor may access only one L2 cache directly and must use the coherence fabric to access other L2 caches. Figure 1 (b) shows five possible partitioning schemes in a 16 processor CMP that have sharing degrees of 1, 2, 4, 8, and 16, respectively. With a sharing degree of 1, there are sixteen 1 MB caches, each of which is private to one processor. With a sharing degree of 16, there is only one 16 MB cache, which is shared by all 16 processors.

To optimize bank organization, we evaluated 5 different bank sizes: 64KB, 128KB, 256KB, 512KB, and 1MB. We estimated bank access latencies using Cacti [18] and a wire delay model [1]. Network hop delays are derived from the dimension of banks, with switching overheads. Among the five bank configurations, our experiments show that the 64KB bank size has the best performance across all experimental configurations. For the remainder of this paper, we assume a 16x16 64KB bank array.

To change sharing degrees, the bank mapping tables in L1 cache controllers, bank controllers, and the central on-chip directory are updated by the operating system. For fully reconfigurable sharing degrees, the sharing status vectors both in L2 tags and the central directory should have enough bits to represent all processors in the CMP. In this substrate, the size of bit vectors in L2 tags and the central directory is the maximum 16.

2.3 Mapping Algorithms

With multiple banks per L2 cache, we have the choice of either always putting a block into a designated bank or allowing a block to reside in one of multiple banks (but not simultaneously). We call the former *static mapping* and the latter *dynamic mapping*.

In static mapping, a fixed hash function uses the lower bits of a block address to select the correct bank. L2 access latency is proportional to the distance from the issuing L1 cache to the L2 cache bank. By allowing non-uniform hit latencies, static mapping can reduce hit latencies of traditional monolithic cache designs, which fix the latency to the longest path [13]. Because a block can be placed into only one bank, its L2 access latency is essentially decided by its address. If frequently accessed blocks happen to map to banks with longer latencies, static mapping will not provide optimal performance.

Dynamic mapping (D-NUCA) addresses the problem faced by static mapping by allowing a block to go to multiple candidate banks, or a *bank set*. With proper placement and migration policies, D-NUCA enables the cache to place frequently accessed blocks in the banks closest to the CPU

and less frequently accessed block in the banks that are farther away. Previous studies have shown that generational promotion that migrates blocks towards banks near the requesting processors yields good performance in uniprocessor systems [5, 13].

CMPs pose new challenges to dynamic mapping. First, migration in multiple directions can cause migration conflicts, with shared blocks ping-ponging between two processors. Second, searching blocks in bank sets becomes more complicated than single-processor D-NUCA organizations. Past studies have shown that centralized partial tags work well in uniprocessor D-NUCA [13]. In CMPs, however, besides increased bandwidth requirements on the central partial tag array, the central partial tags cannot be located close to all processors, thus requiring multi-hop latencies to access the tags.

We address the challenge with two mechanisms: distributed partial tags and a dynamic lookup mechanism.

2.3.1 Migration Policies

A simple migration policy permits a block to be mapped to only one column and restricts migration to the vertical dimension only (D-NUCA 1D, Figure 2 (a)). In this policy, the vertical migration does not reduce the network latency for the horizontal traversal of requests and data blocks. We assumed the following bank set policies for misses: new cache blocks are inserted at the tail of a column bank set. For 8 and 16 sharing degrees, the victim banks are selected from the banks in the middle.

The second migration policy allows blocks to be mapped to any bank without restriction. Migration can happen in both vertical and horizontal directions (D-NUCA 2D, Figure 2 (b)). This policy can further reduce hit latency by decreasing horizontal network latencies. However, it requires a more complicated search mechanism since an L1 miss might need to look up all banks in a shared cache. In D-NUCA 2D, new blocks are inserted at the column closest to the requesting processor.

Unlike single-processor D-NUCA caches, in which blocks are always promoted in one direction, multiple processors from different locations in CMPs can cause cache blocks to be promoted in conflicting directions. In a worst-case scenario, a block may ping-pong between two adjacent banks, with no reduction in hit latencies. To reduce unnecessary migration for conflicting promotion, we simulated two-bit saturated counters embedded in the cache tags, which allow a block to migrate only if the relevant counter for that moving direction is saturated.

2.3.2 Lookup Mechanisms

A partial tag structure replicates low-order bits of full cache tags as a way to reduce the number of requests to full tags [12]. In single-processor D-NUCA caches, centralized partial tags detect L2 misses early and reduce the number of requests to banks. However, the centralized partial tag approach has three drawbacks. First, since the global partial tag array is required to hold the information about all cache blocks in the cache, its access time and energy consumption will be relatively large. Second, since the centralized structure should be placed near the center of the cache, wire delays to and from it can be significant. Finally, the centralized tag array may require many ports since all primary cache misses must access it.

Parameter	Value
Processor frequency	10 GHz
Issue width	4
Window size	64-entry RUU
Number of CPUs	16
L1 I/D cache	32KB, 2-way, 64B block, 8 MSHRs
L2 cache	16x16 banks
L2 cache bank	64KB, 16-way, 5 cycle latency
Network	1 cycle latency between two adjacent banks
On-chip directory	10 cycle access latency
Main Memory	260 cycle latency, 360 GB/s bandwidth

Table 1: Simulated system configuration

Application	Dataset/Parameters
SPECWeb99	Apache web server, file set: 230MB
TPC-W	185MB databases using Apache & MySQL
SPECjbb	IBM JVM version 1.1.8, 16 warehouses
Ocean	258×258 grid
Barnes	16K particles
LU	512×512, 16×16 blocks
Radix	1M integers

Table 2: Workloads.

To overcome these drawbacks, we employ a distributed scheme: partial tags are distributed over the columns, and each column's partial tag array tracks the state of blocks cached in that column. Any changes in a bank column's contents must be reflected in its partial tags synchronously. Cache lookups for a bank column always start from the distributed partial tag array. In the SD=8 or 16 cases, we replicate the partial tags at both ends of a column. This doubles the space overheads of partial tags, but greatly decreases the distance from processors to column partial tags.

In D-NUCA 1D, the search mechanism is straightforward with column partial tags. L1 misses are sent to the head of statically mapped columns, and the first bank and partial tags are accessed simultaneously. For a miss in the first bank, the column partial tags can command further lookups of other banks in the same column or start an L2 refill request. In D-NUCA 2D, the partial tags of all columns that a block can map to may be searched. L1 misses are first sent to the column closest to the requesting processor. If the block is not found in that column, other columns' partial tags are searched.

3. EXPERIMENTAL RESULTS

3.1 Methodology

We evaluated our CMP cache designs using an execution-driven full-system simulator derived from IBM's SimOS-PPC, which uses AIX 4.3.1 as the simulated OS. The core timing model extends the SimpleScalar simulator with multiprocessor support. Table 1 shows a summary of the main architectural parameters. We used three commercial applications: SPECWeb99, TPC-W, and SPECjbb, and four scientific shared-memory benchmarks from the SPLASH-2 suite [24]: Ocean, Barnes, LU, and Radix. Table 2 shows the dataset size and other notable features of each application.

As previously discussed, we model an invalidation-based cache coherence protocol in the CMP. The L2 caches are the points of L1 coherence and maintain sharing vectors for L1 caches. The L2 cache bank array is embedded with a

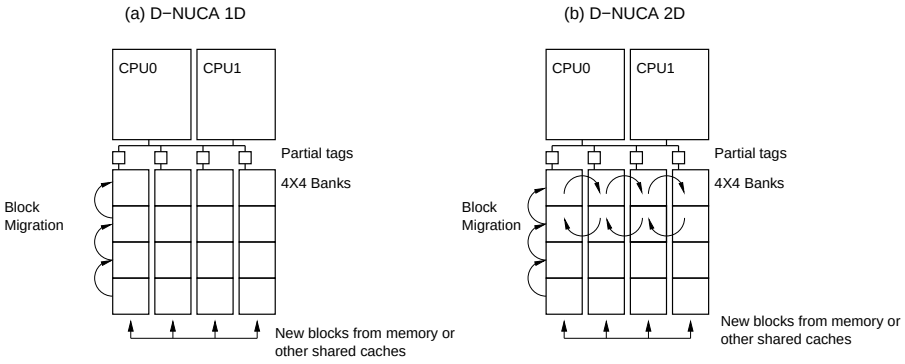

Figure 2: D-NUCA Block Migration Policies

Sharing Degree (SD)	SD=1	SD=2	SD=4	SD=8	SD=16
L2 hit time (cycles)	16.5	18.0	20.9	30.0	35.9
Norm. Remote L2 hits	1.00	0.75	0.60	0.36	n/a
Norm. Mem. accesses	1.00	0.88	0.78	0.78	0.66

Table 3: Average L2 hit times, and normalized remote L2 hits and memory accesses

2D-mesh point-to-point interconnection network comprised of links and switches. All messages for coherence and data migration are modeled to consume network bandwidth; however, we assume infinite buffering at each switching node. A centralized directory is used to track cache lines that are cached in the CMP and is consulted on a cache miss to enforce coherence and/or detect misses.

3.2 Degree of L2 Sharing

In this section, we study the performance effect of the varied L2 cache sharing degrees. In the baseline CMP with S-NUCA based shared caches, the sharing degree changes the effective L2 hit latency, communication frequency among shared caches (e.g., cache-to-cache transfers), and external memory accesses. Table 3 shows the trends in average L2 hit latencies, the number of remote L2 hits and memory accesses as the sharing degree (SD) is changed, with the latter two normalized to the SD=1 case. As the sharing degree increases, shared cache hit latencies increase monotonically from 17 cycles to 36 cycles.

Remote L2 accesses can occur in two cases: First, a read-only shared line is evicted from a local L2 cache, due to a conflict/capacity miss, and upon the next access that line is provided by another shared cache, which has an intact copy of that line. Second, a producer-consumer line is invalidated by another processor in a remote shared cache. In both cases, communication across shared cache boundaries is more expensive in terms of latency and energy consumption than intra-shared cache communication. After detecting a miss in the local shared cache, a miss request is sent to the centralized on-chip directory. The on-chip directory re-transmits the request to another shared cache to service the miss if the cache line is on-chip. Such remote L2 accesses (cache-to-cache transfers) decrease as the sharing degree increases since the likelihood that shared cache lines fall within the same L2 cache increases.

Memory accesses also decrease as the sharing degree increases. With a high sharing degree, processors can share the cache capacity dynamically, reducing the effect of a temporal working set imbalance, resulting in fewer capacity misses than smaller caches. Furthermore, by reducing the number of replicated copies of shared data, the limited on-chip cache capacity can be more efficiently utilized. As shown in Table 3, increasing the sharing degree to 16 can reduce the external memory accesses by 33% on average.

In several ways, the sharing degree affects L1 miss penalties. Figure 3 shows a breakdown of the average L1 miss penalty for each application as the L2 cache sharing degree is varied. L1 misses are served by either the local L2 cache, a remote L2 cache, or external memory. Each bar in the graph shows how much latency each component contributes to the average L1 miss penalty.

Across all benchmark applications, the L2 hit time component (black bar) increases monotonically as the sharing degree increases, due to the wire delay increase in the larger caches. The remote L2 and memory components decrease as the degree of L2 sharing is increased; however, those reductions in latency are often not enough to outweigh the increase in local L2 hit latency. For applications that have high local L2 hit rates and low sharing of cache lines like Barnes-Hut, an increased sharing degree beyond private L2 caches degrades performance.

Figure 4 shows the relative execution times for each application as the sharing degree is varied. As expected from Figure 3, SPECWeb99, Ocean, and LU have the best performance at sharing degree two or four. For Barnes and SPECjbb, increasing the sharing degree degrades performance, with the best performance at the sharing degree of one. The bar with the lowest average L1 miss penalty does not always correspond to the fastest execution time since each miss may affect the execution time with different weight (e.g., SPECWeb99 with SD=4). Critical misses that are not overlapped with other misses can increase execution times more than non-critical misses.

We draw three conclusions from these results. First, building high-degree shared caches for CMPs does not have any advantage in wire-delay dominated future technologies even when high degrees of application sharing exist. The increase in L2 hit latency in shared caches degrades performance more than the reduced misses improve it. Second, the sharing degree can change overall performance significantly. The difference between the best and worst sharing degree ranges from 9% to 22%. Third, no single sharing degree provides the best performance for all the benchmarks. The best performing sharing degree varies across applications. Nevertheless, the SD=4 design point has the best

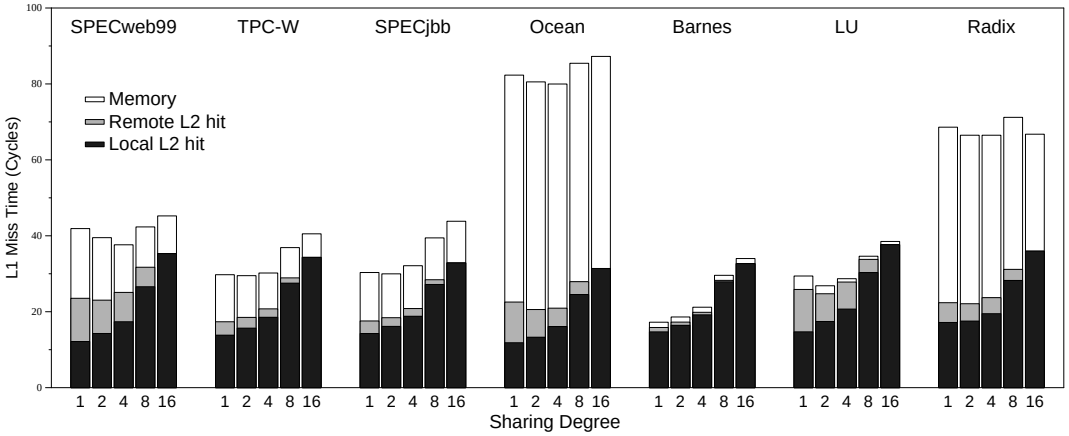

Figure 3: S-NUCA L1 miss latencies (16x16 banks)

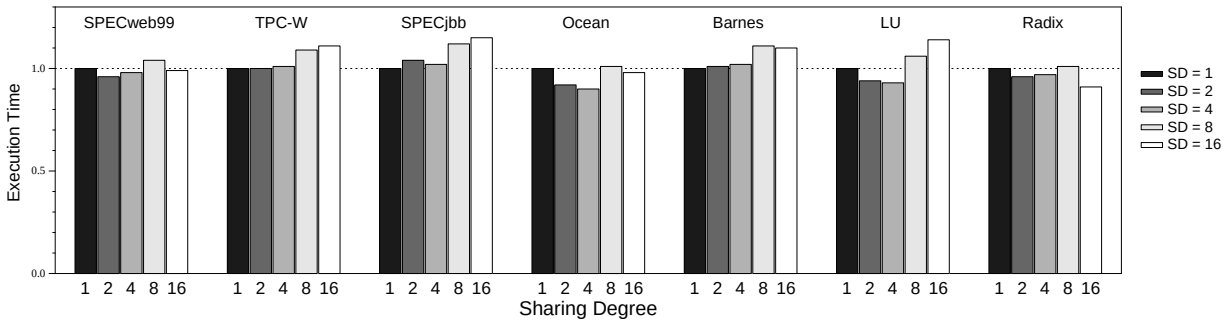

Figure 4: S-NUCA execution time (normalized to SD=1)

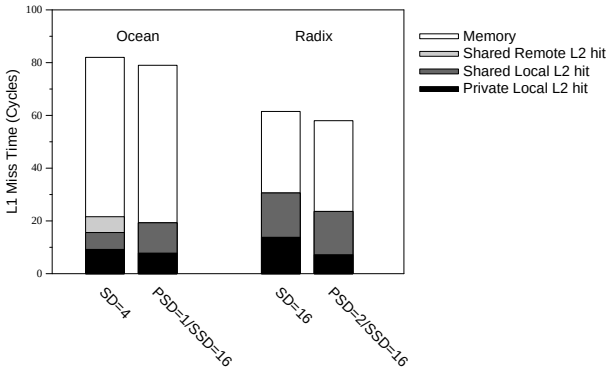

Figure 5: Per-line sharing degrees

average performance for the applications used in this evaluation, and would be the best compromise fixed design point for this mix of workloads for S-NUCA.

3.3 Per-line Sharing Degree

Sharing degree can affect individual cache blocks in different ways based on the sharing patterns of the block. For private blocks, which will never be replicated in other caches, a low sharing degree can reduce the access latencies without hurting caching efficiency. For shared blocks, a higher sharing degree may be more beneficial than lower sharing degree when the reduction of replicated blocks can decrease miss rates significantly. In this section, we investigate the potential benefit of multiple sharing degrees for different classes of blocks.

We divide the address space into private and shared block addresses, and assign different sharing degrees to the two address classes. To evaluate the per-line sharing degrees, we simulated S-NUCA with the two different sharing degrees, *private* and *shared* (PSD and SSD). Since our full-system simulator does not have support for distinguishing private and shared blocks, we used an approximate method of tracking access patterns for the entire address space during run time. Until more than one processor access a block address, the address is assumed to be a private block. Once an address is recognized as a shared block, the block is re-mapped to caches by shared sharing degree. We assumed that the shared sharing degree is always higher than or equal to the private sharing degree.

Figure 5 presents the breakdown of L1 miss penalties for the two applications in which per-line sharing degree is effective. We measured all combinations of private and shared sharing degrees. The two bars for each application represent the baseline ideal uniform sharing degree and the best per-line sharing degree configuration. We divided the local L2 hit latencies into private and shared accesses.

For the two applications, per-line sharing degree reduced execution time by 7% and 6%. With fixed sharing degree, the best sharing degree was 4 (Ocean) or 16 (Radix). When different sharing degrees are allowed for private and shared blocks, the best combination is 1 or 2 for private sharing degree and 16 for shared sharing degree. For these two

Sharing Degree	SD=1	SD=2	SD=4	SD=8	SD=16
S-NUCA	16.5	18.0	20.9	30.0	35.9
D-NUCA 1D Perfect	9.8	11.2	13.8	21.7	28.6
D-NUCA 1D Real	11.2	12.3	15.0	24.8	31.5
D-NUCA 2D Perfect	10.0	10.4	11.7	19.8	25.2
D-NUCA 2D Real	11.6	13.5	16.2	25.1	31.9

Table 4: Average D-NUCA L2 hit latencies

benchmarks, access latencies to private blocks are reduced by having lower sharing degrees (1 or 2), while replications are minimized with high shared a sharing degree (16). With a low private sharing degree, private blocks, which will never be accessed by other processors, are placed on a small range of banks close to processors. With a high shared sharing degree, block replication is reduced, although the shared blocks may be spread across a larger range of banks than private blocks, thus increasing access latencies. It is possible that a finer-grained distribution of lines to banks would improve performance.

3.4 Overcoming Wire Delay Dominance

In Section 3.2, we showed the performance trade-offs of changing the sharing degree in CMP S-NUCA L2 caches. In this section, we evaluate dynamic mapping policies which can potentially reduce long latencies with large sharing degrees. Performance improvements are achieved when the migration policy is successful and the reduction in latency dominates the increased latency of the more complex lookup mechanism. To isolate the effectiveness of dynamic migration from the overheads of the search mechanism, we evaluated two more configurations: perfect D-NUCA 1D and 2D caches. The two configurations assume an oracle searching mechanism that allows L1 misses to be sent directly to the L2 bank storing the requested block on a hit. L2 misses are detected without any overhead. The simulated system models other overheads such as network and bank bandwidth consumption for accesses and migration in detail.

Table 4 shows the average L2 hit times across all applications for five sharing degrees. With the perfect lookup mechanism, both 1D and 2D migration policies show significant reductions in L2 hit latencies. The latency reductions increase as the sharing degree increases. At SD=16, the perfect D-NUCA 1D and 2D policies reduce the average L2 hit latency by 22% and 33%, respectively compared to the S-NUCA design. However, with a realistic search mechanism with distributed partial tags, the latency reduction of D-NUCA is significantly reduced, confirming the search mechanism is a key design issue with D-NUCA.

Figure 6 shows the relative execution times of the best performing sharing degree for the S-NUCA and D-NUCA design points across all applications. Each bar shows the SD with the best performance noted at the top. This figure illustrates two issues: (1) the performance potential of the two perfect search and migration mechanisms (1D and 2D perfect) and how closely the realistic implementations can match them, and (2) performance of two realistic D-NUCA designs compared to S-NUCA with the best sharing degree.

The perfect search mechanisms with 1D and 2D dynamic migration can reduce execution time by 3-25%, except Ocean. For Ocean, although D-NUCA reduces average hit latencies, L2 miss rates are increased since blocks are not promoted quickly, and thus replaced by new blocks. For SPECjbb, the performance improvement is small, since SPECjbb does not take any advantage of the increased sharing degree, and the effect of dynamic migration is not high at low sharing degrees.

With realistic search mechanisms, performance improvement of D-NUCA can be lost (SPECWeb99 and TPC-W). For LU and Radix, both 1D and 2D migrations show large improvement by 17%-20%. LU has a relatively large L1 data miss rate of 12%, but the entire working set nearly fits in the L2 caches. The reduction in L2 hit latencies directly improves performance. In Radix however, external memory accesses dominate performance due to both capacity and conflict misses. The increased bank associativity with dynamic migration reduced conflict misses significantly. Since shared caches, especially with high sharing degrees, are prone to conflict misses, the increased associativity helps avoid certain pathological conflict miss cases.

Although dynamic migration improves the performance of shared caches, the improvement is still modest (less than 5%) for 5 tested applications. Considering the complexity of a D-NUCA implementation and the extra energy consumption due to lookups, it is unlikely that implementing dynamic migration is justified for CMPs.

3.5 Interaction Between L1 Prefetching and NUCA Design Alternatives

In this section, we investigate the effect of hardware-based strided prefetching [2, 10] on NUCA design alternatives. Since effective prefetching can tolerate L1 miss latencies, it can potentially diminish the effect of the increased L2 hit latency observed with larger sharing degrees. We evaluated the effect of strided prefetching on the CMP caches using an implementation similar to the one used by Beckmann et al [4], but restricted to L1 prefetching. The strided prefetching strategy uses three filters with 32 entries each to detect streams. The three filters, positive unit stride, negative unit stride, and non-unit stride use four consecutive misses before confirming a stream, and allocate an entry in an eight-entry stream table. As soon as a stream is detected, six consecutive prefetch requests are issued on behalf of the L1 caches. Prefetching can be stopped when all MSHR entries are used or prefetches cross physical page boundaries. Prefetched cache blocks are stored directly into the caches, which may cause cache pollution problems. If the processors access a prefetched block, another prefetch for the stream is issued from the stream table if the entry is still resident. Stream table entries are replaced using an LRU policy.

Table 5 shows the coverage and accuracy of L1 prefetching. *Coverage* is the ratio of prefetch hits to L1 misses, and *accuracy* is the ratio of prefetch hits to the total number of prefetches, ignoring any late prefetches that may partially hide latency. SPECjbb and LU have small coverage (0.5% for SPECjbb and 0.1% for LU) and relatively low accuracy. Prefetching is most effective for Ocean, with a coverage of 32% of the L1 data misses and 95% accuracy.

Figure 7 shows the relative execution times of S-NUCA and D-NUCA 2D with L1 prefetching compared to the baseline without prefetching using the best performing SD configuration for each run. For S-NUCA shared caches, L1 prefetching can reduce execution time for SPECweb99 (3%), Ocean (10%), and Radix (20%). For SPECjbb, L1 prefetching does not reduce execution time due to the low coverage. Although prefetching can improve performance for many applications for the different configurations, it does not change

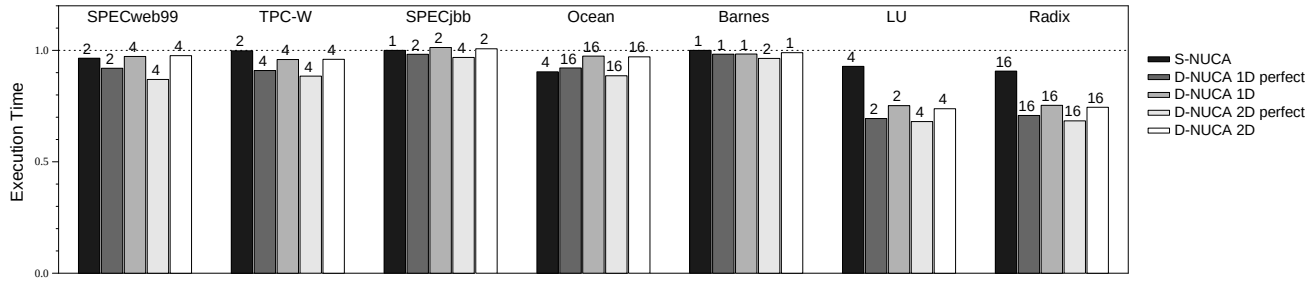

Figure 6: D-NUCA execution time (normalized to S-NUCA with SD=1)

	SPECweb99	TPC-W	SPECjbb	Ocean	Barnes	LU	Radix
L1 I Coverage	31.3%	14.1%	28.3%	20.3%	15.0%	14.8%	25.5%
L1 I Accuracy	46.4%	39.2%	34.0%	61.4%	50.4%	56.5%	49.4%
L1 D Coverage	14.0%	6.8%	0.5%	32.4%	12.2%	0.1%	10.3%
L1 D Accuracy	96.9%	90.0%	35.2%	95.3%	41.5%	0.1%	98.8%

Table 5: Prefetching coverage and accuracy for L1 instruction and data caches

the choice of the best average sharing degree for each design significantly. For six applications, the best sharing degrees with prefetching are either the same as or close to the best sharing degree without L1 prefetching.

We observe that prefetching can also improve dynamic migration. For applications where S-NUCA prefetching is effective, D-NUCA caches have similar performance improvements. This observation confirms that dynamic migration and prefetching are complementary memory latency reduction/tolerance techniques.

3.6 Energy Trade-Offs

One concern with dynamic migration designs is the increased energy consumption due to the complex search mechanism and cache line movement. Instead of estimating the actual energy consumption, we indirectly show the total number of bank accesses of S-NUCA and 2D D-NUCA. For each application, we compared the best sharing degree of the two configurations in Table 6. The numbers in parentheses in the 2D D-NUCA row show the percentage of extra bank accesses compared to the S-NUCA case.

As expected, block migration increases the total bank accesses significantly. The extra bank accesses for block migration constitute 28-48% for the tested applications. In shared caches, unnecessary block migration may occur more frequently than private D-NUCA caches. Although we reduced the unnecessary migration by 2-bit saturating counters, the number of bank accesses due to migration is still significant.

D-NUCA lookup mechanisms also consume energy. For each L2 access, at least a part of distributed partial tags should be accessed. With D-NUCA 2D, the number of partial tag lookups may increase, if blocks are not found in the closest column.

4. RELATED WORK

Shared Cache Designs: Shared caches have been studied in the context of chip multiprocessors and multithreaded processors. Sohi and Franklin's early study proposed the interleaved banks for extra ports in private L1 caches, which resembles multi-banking in shared caches [19]. Nayfeh et al. investigated shared caches for primary and secondary caches on a multi-chip module substrate with four CPUs [16]. They examined how the memory sharing patterns of different applications affect the best cache hierarchy. Subsequent work from the same authors showed the trade-offs of shared-cache clustering in multi-chip multiprocessors [17]. With eight CPUs, they observed for private L2 caches, a coherence bus becomes the performance bottleneck, suggesting shared caches to reduce the bus traffic.

Recent studies considered wire latency as a primary design factor in CMP caches. Beckmann et al. compared three latency reduction techniques including D-NUCA for CMPs with an 8-CPU shared cache [4]. Their study fixed the sharing degree to 8 and observed that combining the three latency reduction techniques can decrease the L2 hit latencies of CMPs. With NuRAPID-based CMP L2 designs, Chishti et al. studied optimizations to reduce unnecessary replication and communication overheads [6]. Speight et al. studied how CMP L2 caches interact with off-chip L3 caches and how on-chip L2 caches temporarily absorb modified replacement blocks from other caches [20].

Dynamic Partitioning: Several researches investigated dynamically re-allocating cache capacity for CMPs and multithreaded processors. Suh et al. studied a monitoring system for an optimal dynamic partitioning [21], and a hardware partitioning mechanism for set-associative caches [22]. Liu et al. proposed achieving dynamic bank allocation by re-mapping the banks [15]. Iyer proposed a priority-based cache management systems to allocate cache resources by OS-assigned priority [9]. To prevent thread starvation due to cache capacity sharing, Kim et al. investigated fairness issues in CMP cache sharing [14].

5. CONCLUSION

The CMP organization that we introduced in this paper is designed to support both low-latency, private logical caches as well as highly shared caches, simply by adjusting the mapping of the same address on different processors to the L2 cache.

The results showed that–compared to private, non-shared L2 partitions–the L2 latency more than doubled for a fully shared cache. The results also showed that the fully shared

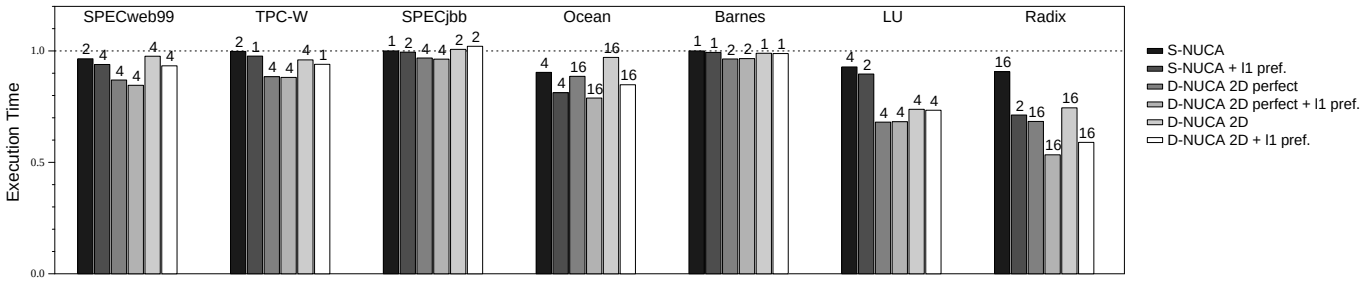

Figure 7: Prefetching execution time (normalized to S-NUCA with SD=1)

	SPECweb99	TPC-W	SPECjbb	Ocean	Barnes	LU	Radix
S-NUCA Best	44	25	4	10	12	31	20
D-NUCA 2D Best	72 (40%)	34 (28%)	6 (32%)	18 (49%)	15 (27%)	60 (43%)	49 (48%)

Table 6: Bank accesses per 1K instructions

cache could eliminate a third of off-chip misses. Clearly, a large opportunity exists if this gap can be bridged.

The S-NUCA organization worked best with a low-to-medium sharing degree for all applications; the extra hit latency was simply too detrimental with larger sharing degrees. Consequently, we evaluated L1 prefetching and dynamic migration of lines, attempting to reduce the average hit latencies and make the larger sharing degrees more effective. L1 prefetching worked uniformly well, but did not drive the ideal sharing degree significantly in one direction or the other, even though the L1 miss rates were reduced.

Dynamic migration (D-NUCA 1D and 2D) showed modest performance improvements, despite reductions in average hit latency. However, for a subset of applications, D-NUCA drove the ideal sharing degree to higher sharing degrees, showing that mechanisms to reduce latency can indeed make higher-degree shared caches the optimal point. It remains to be seen whether the added complexity and power consumption justify moving to a D-NUCA design since only a subset of the applications benefit appreciably; we think it unlikely to be justifiable.

Since the underlying cache framework permits different degrees of sharing on the same hardware, further opportunity exists: The cache can be configured differently to have the ideal sharing degree for each specific application or for individual cache lines. Figure 8 shows a comparison of the average execution time across all applications for the S-NUCA and D-NUCA designs normalized to the S-NUCA design with the best sharing degree of four. For each policy, we show the performance with the best fixed sharing degree across all applications, the worst fixed sharing degree, and a per-application "variable" degree. Choosing the best sharing degree at a finer granularity provides a small but measurable (5%) speedup for the more aggressive policies. In Table 7, we list the ideal per-application sharing degrees for each policy and show the percentage speedup over the best fixed sharing degree for that policy. For D-NUCA, SPECjbb, Ocean, and Radix showed a measurable boost of 5-25%.

Based on these results, we conclude that the simplest design is probably the best: an S-NUCA organization with a sharing degree of two or four. However, the D-NUCA results still hold promise, and we are continuing to explore ways to exploit the flexible mapping. Treating different classes of lines with different sharing degrees showed significant potential for two applications, and this approach is a subject of continuing effort.

6. ACKNOWLEDGMENTS

This research is supported by the Defense Advanced Research Projects Agency (DARPA) under contracts F33615-01-C-1892 and NBCH30390004, NSF infrastructure grant EIA-0303609, and two IBM University Partnership Awards. We thank the members of the IBM PERCS team for their technical feedback.

7. REFERENCES

[1] V. Agarwal, S. W. Keckler, and D. Burger. The effect of technology scaling on microarchitecture structures. Technical Report TR-00-02, Department of Computer Sciences, University of Texas at Austin, May 2001.

[2] J.-L. Baer and T.-F. Chen. Effective hardware-based data prefetching for high-performance processors. *IEEE Transactions on Computer*, 44(5):609–623, 1995.

[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 *The 27th Annual International Symposium on Computer Architecture*, pages 282–293, June 2000.

[4] B. M. Beckmann and D. A. Wood. Managing wire delay in large chip-multiprocessor caches. In *37th International Symposium on Microarchitecture (MICRO)*, December 2004.

[5] Z. Chishti, M. Powell, and T. N. Vijaykumar. Distance associativity for high-performance energy-efficient non-uniform cache architectures. In *The 36th Annual International Symposium on Microarchitecture (MICRO)*, pages 55–66, December 2003.

[6] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Optimizing replication, communication, and capacity allocation in cmps. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, 2005.

[7] L. Hammond, B. A. Hubbert, M. Siu, M. K. Prabhu, M. Chen, and K. Olukotun. The Stanford Hydra

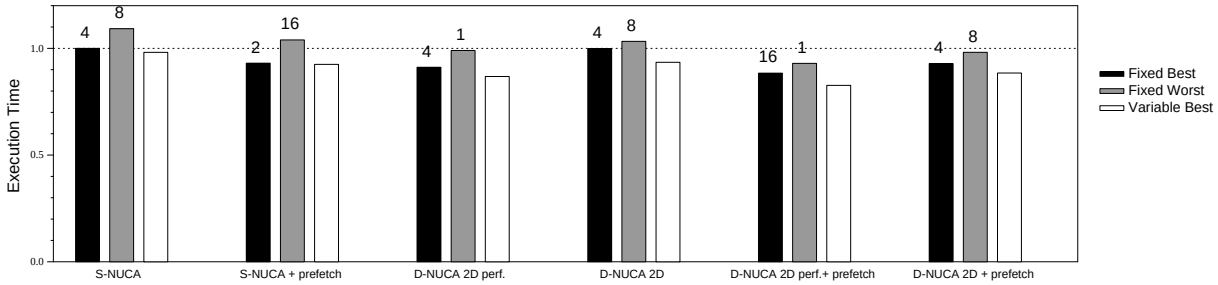

Figure 8: Execution time for fixed best, fixed worst and variable best sharing degree

	SPECweb99	TPC-W	SPECjbb	Ocean	Barnes	LU	Radix	Fixed Best
S-NUCA	2 (1.4%)	2 (1.4%)	1 (1.6%)	4 (0.0%)	1 (2.0%)	4 (0.0%)	16 (6.6%)	4
S-NUCA +pref.	4 (0.4%)	1 (0.4%)	2 (0.0%)	4 (2.4%)	1 (0.9%)	2 (0.0%)	2 (0.0%)	2
D-NUCA 2D	4 (0.0%)	4 (0.0%)	2 (9.4%)	16 (4.1%)	1 (3.6%)	4 (0.0%)	16 (25.4%)	4

Table 7: Per-application best sharing degrees

CMP. *IEEE Micro*, pages 71–84, December 2000.

[8] J. Huh, J. Chang, D. Burger, and G. S. Sohi. Coherence decoupling: Making use of incoherence. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2004.

[9] R. Iyer. CQoS: a framework for enabling QoS in shared caches of cmp platforms. In *Proceedings of the 18th annual international conference on Supercomputing*, pages 257–266, 2004.

[10] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th annual international symposium on Computer Architecture*, pages 364–373, 1990.

[11] R. Kalla, B. Sinharoy, and J. M. Tendler. IBM Power5 Chip: A dual-core multithreaded processor. *IEEE Micro*, 24(2), Mar/Apr 2004.

[12] R. Kessler, R. Jooss, A. Lebeck, and M. Hill. Inexpensive implementations of set-associativity. In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, pages 131–139, May 1989.

[13] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 211–222, October 2002.

[14] S. Kim, D. Chandra, and Y. Solihin. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *Proceedings of the 13th International Conference on Parallel Architecture and Compilation Techniques (PACT'04)*, pages 111–122, 2004.

[15] C. Liu, A. Sivasubramaniam, and M. Kandemir. Organizing the last line of defense before hitting the memory wall for cmps. In *Proceedings of the 10th International Symposium High Performance Computer Architecture*, Feb. 2004.

[16] B. A. Nayfeh, L. Hammond, and K. Olukotun. Evaluation of design alternatives for a multiprocessor microprocessor. In *Proceedings of the 23th Annual International Symposium on Computer Architecture*, pages 67–77, May 1996.

[17] B. A. Nayfeh, K. Olukotun, and J. P. Singh. The impact of shared-cache clustering in small-scale shared-memory multiprocessors. In *Proceedings of the 2nd IEEE Symposium on High-Performance Computer Architecture (HPCA)*, page 74, 1996.

[18] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. Technical Report 2001-2, HP, Western Research Laboratory, 2001.

[19] G. Sohi and M. Franklin. High-performance data memory systems for superscalar processors. In *Proceedings of the Fourth Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 53–62, Apr. 1991.

[20] E. Speight, H. Shafi, L. Zhang, and R. Rajamony. Adaptive mechanisms and policies for managing cache hierarchies in chip multiprocessors. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, 2005.

[21] G. Suh, S. Devadas, and L. Rudolph. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *Proceedings of the 8th International Symposium High Performance Computer Architecture*, Feb. 2002.

[22] G. E. Suh, L. Rudolph, and S. Devadas. Dynamic partitioning of shared cache memory. *Journal of Supercomputing*, 28(1):7–26, 2004.

[23] J. M. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy. Power4 system microarchitecture. *IBM Journal of Research and Development*, 46(1), 2002.

[24] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The splash-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 24–36, 1995.