Managing Wire Delay in Large Chip-Multiprocessor Caches

Bradford M. Beckmann and David A. Wood
Computer Sciences Department
University of Wisconsin—Madison
{beckmann, david}@cs.wisc.edu

Abstract

In response to increasing (relative) wire delay, architects have proposed various technologies to manage the impact of slow wires on large uniprocessor L2 caches. Block migration (e.g., D-NUCA [27] and NuRapid [12]) reduces average hit latency by migrating frequently used blocks towards the lower-latency banks. Transmission Line Caches (TLC) [6] use on-chip transmission lines to provide low latency to all banks. Traditional stride-based hardware prefetching strives to tolerate, rather than reduce, latency.

Chip multiprocessors (CMPs) present additional challenges. First, CMPs often share the on-chip L2 cache, requiring multiple ports to provide sufficient bandwidth. Second, multiple threads mean multiple working sets, which compete for limited on-chip storage. Third, sharing code and data interferes with block migration, since one processor's low-latency bank is another processor's high-latency bank.

In this paper, we develop L2 cache designs for CMPs that incorporate these three latency management techniques. We use detailed full-system simulation to analyze the performance trade-offs for both commercial and scientific workloads. First, we demonstrate that block migration is less effective for CMPs because 40-60% of L2 cache hits in commercial workloads are satisfied in the central banks, which are equally far from all processors. Second, we observe that although transmission lines provide low latency, contention for their restricted bandwidth limits their performance. Third, we show stride-based prefetching between L1 and L2 caches alone improves performance by at least as much as the other two techniques. Finally, we present a hybrid design—combining all three techniques—that improves performance by an additional 2% to 19% over prefetching alone.

1 Introduction

Many factors—both technological and marketing—are driving the semiconductor industry to implement multiple processors per chip. Small-scale chip multiprocessors (CMPs), with two processors per chip, are already commercially available [24, 30, 44]. Larger-scale CMPs seem likely to follow as transistor densities increase [5, 18, 45, 28]. Due to the benefits of sharing, current and future CMPs are likely to have a shared, unified L2 cache [25, 37].

Wire delay plays an increasingly significant role in cache design. Design partitioning, along with the integration of more metal layers, allows wire dimensions to decrease slower than transistor dimensions, thus keeping wire delay controllable for short distances [20, 42]. For instance as technology improves, designers split caches into multiple banks, controlling the wire delay within a bank. However, wire delay between banks is a growing performance bottleneck. For example, transmitting data 1 cm requires only 2-3 cycles in current (2004) technology, but will necessitate over 12 cycles in 2010 technology assuming a cycle time of 12 fanout-of-three delays [16]. Thus, L2 caches are likely to have hit latencies in the tens of cycles.

Increasing wire delay makes it difficult to provide uniform access latencies to all L2 cache banks. One alternative is Non-Uniform Cache Architecture (NUCA) designs [27], which allow nearer cache banks to have lower access latencies than further banks. However, supporting multiple processors (e.g., 8) places additional demands on NUCA cache designs. First, simple geometry dictates that eight regular-shaped processors must be physically distributed across the 2-dimensional die. A cache bank that is physically close to one processor cannot be physically close to all the others. Second, an 8-way CMP requires eight times the sustained cache bandwidth. These two factors strongly suggest a physically distributed, multi-port NUCA cache design.

This paper examines three techniques—previously evaluated only for uniprocessors—for managing L2 cache latency in an eight-processor CMP. First, we consider using hardware-directed stride-based prefetching [9, 13, 23] to tolerate the variable latency in a NUCA cache design. While current systems perform hardware-directed strided prefetching [19, 21, 43], its effectiveness is workload dependent [10, 22, 46, 49]. Second, we consider cache block migration [12, 27], a recently proposed technique for NUCA caches that moves frequently accessed blocks to cache banks closer to the requesting processor. While block migration works well for uniprocessors, adapting it to

This work was supported by the National Science Foundation (CDA-9623632, EIA-9971256, EIA-0205286, and CCR-0324878), a Wisconsin Romnes Fellowship (Wood), and donations from Intel Corp. and Sun Microsystems, Inc. Dr. Wood has a significant financial interest in Sun Microsystems, Inc.

CMPs poses two problems. One, blocks shared by multiple processors are pulled in multiple directions and tend to congregate in banks that are equally far from all processors. Two, due to the extra freedom of movement, the effectiveness of block migration in a shared CMP cache is more dependent on "smart searches" [27] than its uniprocessor counterpart, yet smart searches are harder to implement in a CMP environment. Finally, we consider using on-chip transmission lines [8] to provide fast access to all cache banks [6]. On-chip transmission lines use thick global wires to reduce communication latency by an order of magnitude versus long conventional wires. Transmission Line Caches (TLCs) provide fast, nearly uniform, access latencies. However, the limited bandwidth of transmission lines—due to their large dimensions—may lead to a performance bottleneck in CMPs.

This paper evaluates these three techniques—against a baseline NUCA design with L2 miss prefetching—using detailed full-system simulation and both commercial and scientific workloads. We make the following contributions:

- Block migration is less effective for CMPs than previous results have shown for uniprocessors. Even with an perfect search mechanism, block migration alone only improves performance by an average of 3%. This is in part because shared blocks migrate to the middle equally-distant cache banks, accounting for 40-60% of L2 hits for the commercial workloads.
- Transmission line caches in CMPs exhibit performance improvements comparable to previously published uniprocessor results [6]—8% on average. However, contention for their limited bandwidth accounts for 26% of L2 hit latency.
- Hardware-directed strided prefetching hides L2 hit latency about as well as block migration and transmission lines reduce it. However, prefetching is largely orthogonal, permitting hybrid techniques.
- A hybrid implementation—combining block migration, transmission lines, and on-chip prefetching—provides the best performance. The hybrid design improves performance by an additional 2% to 19% over the baseline.
- Finally, prefetching and block migration improve network efficiency for some scientific workloads, while transmission lines potentially improve efficiency across all workloads.

2 Managing CMP Cache Latency

This section describes the baseline CMP design for this study and how we adapt the three latency management techniques to this framework.

2.1 Baseline CMP Design

We target eight-processor CMP chip designs assuming the 45 nm technology generation projected in 2010 [16]. Table 1 specifies the system parameters for all designs. Each CMP design assumes approximately 300 mm^2 of available die area [16]. We estimate eight 4-wide superscalar processors would occupy 120 mm^2 [29] and 16 MB of L2 cache storage would occupy 64 mm^2 [16]. The on-chip interconnection network and other miscellaneous structures occupy the remaining area.

As illustrated in Figure 1, the baseline design—denoted CMP-SNUCA—assumes a Non-Uniform Cache Architecture (NUCA) L2 cache, derived from Kim, et al.'s S-NUCA-2 design [27]. Similar to the original proposal, CMP-SNUCA statically partitions the address space across cache banks, which are connected via a 2D mesh interconnection network. CMP-SNUCA differs from the uniprocessor design in several important ways. First, it places eight processors around the perimeter of the L2 cache, effectively creating eight distributed access locations rather than a single centralized location. Second, the 16 MB L2 storage array is partitioned into 256 banks to control bank access latency [1] and to provide sufficient bandwidth to support up to 128 simultaneous on-chip processor requests. Third, CMP-SNUCA connects four banks to each switch and expands the link width to 32 bytes. The wider CMP-SNUCA network provides the additional bandwidth needed by an 8-processor CMP, but requires longer latencies as compared to the originally proposed uniprocessor network. Fourth, shared CMP caches are subject to contention from different processors' working sets [32], motivating 16-way set-associative banks with a pseudo-LRU replacement policy [40]. Finally, we assume an idealized off-chip communication controller to provide consistent off-chip latency for all processors.

2.2 Strided Prefetching

Strided or stride-based prefetchers utilize repeatable memory access patterns to tolerate cache miss latency [11, 23, 38]. Though the L1 cache filters many memory requests, L1 and L2 misses often show repetitive access patterns. Most current prefetchers utilize miss patterns to predict cache misses before they happen [19, 21, 43]. Specifically, current hardware prefetchers observe the stride between two recent cache misses, then verify the stride using subsequent misses. Once the prefetcher reaches a threshold of fixed strided misses, it launches a series of fill requests to reduce or eliminate additional miss latency.

We base our prefetching strategy on the IBM Power 4 implementation [43] with some slight modifications. We evaluate both L2 prefetching (i.e., between the L2 cache and memory) and L1 prefetching (i.e., between the L1 and L2 caches). Both the L1 and L2 prefetchers contain three separate 32-entry filter tables: positive unit stride, negative unit stride, and non-unit stride. Similar to Power 4, once a filter table entry recognizes 4 fixed-stride misses, the prefetcher allocates the miss stream into its 8-entry stream table. Upon allocation, the L1I and L1D prefetchers launch 6 consecutive

Table 1. 2010 System Parameters

Memory System		Dynamically Scheduled Processor	
split L1 I & D caches	64 KB, 2-way, 3 cycles	clock frequency	10 GHz
unified L2 cache	16 MB, 256x64 KB, 16-way, 6 cycle bank access	reorder buffer / scheduler	128 / 64 entries
L1/L2 cache block size	64 Bytes	pipeline width	4-wide fetch & issue
memory latency	260 cycles	pipeline stages	30
memory bandwidth	320 GB/s	direct branch predictor	3.5 KB YAGS
memory size	4 GB of DRAM	return address stack	64 entries
outstanding memory requests/CPU	16	indirect branch predictor	256 entries (cascaded)

prefetches along the stream to compensate for the L1 to L2 latency, while the L2 prefetcher launches 25 prefetches. Each prefetcher issues prefetches for both loads and stores because, unlike the Power 4, our simulated machine uses an L1 write-allocate protocol supporting sequential consistency. Also we model separate L2 prefetchers per processor, rather than a single shared prefetcher. We found that with a shared prefetcher, interference between the different processors' miss streams significantly disrupts the prefetching accuracy and coverage[1].

2.3 Block Migration

Block migration reduces global wire delay from L2 hit latency by moving frequently accessed cache blocks closer to the requesting processor. Migrating data to reduce latency has been extensively studied in multiple-chip multiprocessors [7, 15, 17, 36, 41]. Kim, et al. recently applied data migration to reduce latency inside future aggressively-banked uniprocessor caches [27]. Their Dynamic NUCA (D-NUCA) design used a 2-dimensional mesh to interconnect 2-way set-associative banks, and dynamically migrated frequently accessed blocks to the closest banks. NuRapid used centralized tags and a level of indirection to decouple data placement from set indexing, thus reducing conflicts in the nearest banks [12]. Both D-NUCA and NuRapid assumed a single processor chip accessing the L2 cache network from a single location.

For CMPs, we examine a block migration scheme as an extension to our baseline CMP-SNUCA design. Similar to the uniprocessor D-NUCA design [27], CMP-DNUCA permits block migration by *logically* separating the L2 cache banks into 16 unique banksets, where an address maps to a bankset and can reside within any one bank of the bankset. CMP-DNUCA *physically* separates the cache banks into 16 different *bankclusters*, shown as the shaded "Tetris" pieces in Figure 1. Each bankcluster contains one bank from every bankset, similar to the uniprocessor "fair mapping" policy

1. Similar to separating branch predictor histories per thread [39], separating the L2 miss streams by processor significantly improves prefetcher performance (up to 14 times for the workload ocean).

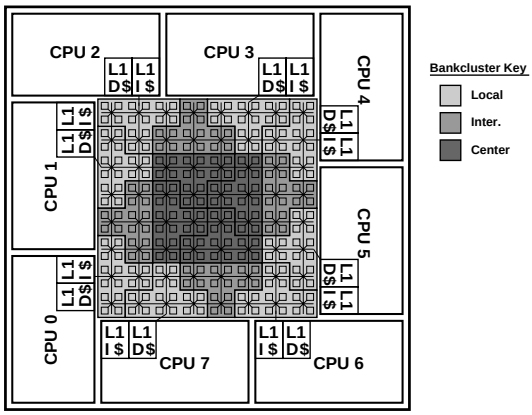

Figure 1. CMP-SNUCA Layout with CMP-DNUCA Bankcluster Regions

[27]. The bankclusters are grouped into three distinct regions. The 8 banksets closest to each processor form the *local* regions, shown by the 8 lightly shaded bankclusters in Figure 1. The 4 bankclusters that reside in the center of the shared cache form the *center* region, shown by the 4 darkest shaded bankclusters in Figure 1. The remaining 4 bankclusters form the *inter*, or intermediate, region. Ideally block migration would maximize L2 hits within each processor's local bankcluster where the uncontended L2 hit latency (i.e., load-to-use latency) varies between 13 to 17 cycles and limit the hits to another processor's local bankcluster, where the uncontended latency can be as high as 65 cycles.

To reduce the latency of detecting a cache miss, the uniprocessor D-NUCA design utilized a "smart search" [27] mechanism using a partial tag array. The centrally-located partial tag structure [26] replicated the low-order bits of each bank's cache tags. If a request missed in the partial tag structure, the block was guaranteed not to be in the cache. This smart search mechanism allowed nearly all cache misses to be detected without searching the entire bankset.

In CMP-DNUCA, adopting a partial tag structure appears impractical. A centralized partial tag structure cannot be quickly accessed by all processors due to wire delays. Fully replicated 6-bit partial tag structures (as used in uni-

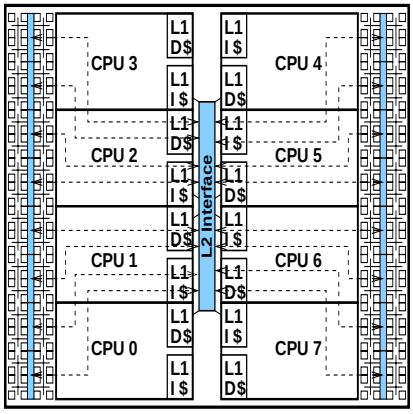

Figure 2. CMP-TLC Layout

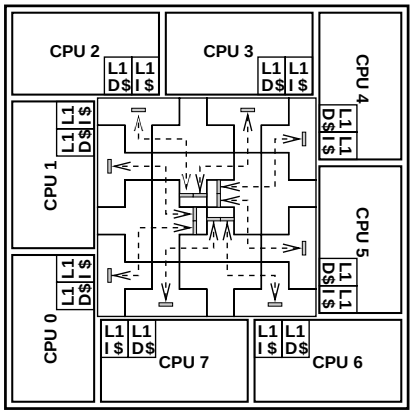

Figure 3. CMP-Hybrid Layout

processor D-NUCA [27]) require 1.5 MBs of state, an extremely high overhead. More importantly, separate partial tag structures require a complex coherence scheme that updates address location state in the partial tags with block migrations. However, because architects may invent a solution to this problem, we evaluate CMP-DNUCA both with and without a perfect search mechanism.

2.4 On-chip Transmission Lines

On-chip transmission line technology reduces L2 cache access latency by replacing slow conventional wires with ultra-fast transmission lines [6]. The delay in conventional wires is dominated by a wire's resistance-capacitance product, or RC delay. RC delay increases with improving technology as wires become thinner to match the smaller feature sizes below. Specifically, wire resistance increases due to the smaller cross-sectional area and sidewall capacitance increases due to the greater surface area exposed to adjacent wires. On the other hand, transmission lines attain significant performance benefit by increasing wire dimensions to the point where the inductance-capacitance product (LC delay) determines delay [8]. In the LC range, data can be communicated by propagating an incident wave across the transmission line instead of charging the capacitance across a series of wire segments. While techniques such as low-k intermetal dielectrics, additional metal layers, and more repeaters across a link, will mitigate RC wire latency for short and intermediate links, transmitting data 1 cm will require more than 12 cycles in 2010 technology [16]. In contrast, on-chip transmission lines implemented in 2010 technology will transmit data 1 cm in less than a single cycle [6].

While on-chip transmission lines achieve significant latency reduction, they sacrifice substantial bandwidth or require considerable manufacturing cost. To achieve transmission line signalling, on-chip wire dimensions and spacing must be an order of magnitude larger than minimum pitch global wires. To attain these large dimensions, transmission lines must be implemented in the chip's uppermost metal layers. The sparseness of these upper layers severely limits the number of transmission lines available. Alternatively, extra metal layers may be integrated to the manufacturing process, but each new metal layer adds about a day of manufacturing time, increasing wafer cost by hundreds of dollars [47].

Applying on-chip transmission lines to reduce the access latency of a shared L2 cache requires efficient utilization of their limited bandwidth. Similar to our uniprocessor TLC designs [6], we first propose using transmission lines to connect processors with a shared L2 cache through a single L2 interface, as shown in Figure 2. Because transmission lines do not require repeaters, CMP-TLC creates a direct connection between the centrally located L2 interface and the peripherally located storage arrays by routing directly over the processors. Similar to CMP-SNUCA, CMP-TLC statically partitions the address space across all L2 cache banks. Sixteen banks (2 adjacent groups of 8 banks) share a common pair of thin 8-byte wide unidirectional transmission line links to the L2 cache interface. To mitigate the contention for the thin transmission line links, our CMP-TLC design provides 16 separate links to different segments of the L2 cache. Also to further reduce contention, the CMP-TLC L2 interface provides a higher bandwidth connection (80-byte wide) between the transmission lines and processors than the original uniprocessor TLC design. Due to the higher bandwidth, requests encounter greater communication latency (2-10 cycles) within the L2 cache interface.

We also propose using transmission lines to quickly access the central banks in the CMP-DNUCA design. We refer to this design as CMP-Hybrid. CMP-Hybrid, illustrated in Figure 3, assumes the same design as CMP-DNUCA except the closest switch to each processor has a 32-byte wide transmission line link to a center switch in the DNUCA cache. Because the processors are distributed around the perimeter of the chip and the distance between the processor switches and the center switches is relatively short (approximately 8 mm), the transmission line links in CMP-Hybrid are wider (32 bytes) than their CMP-TLC counterparts

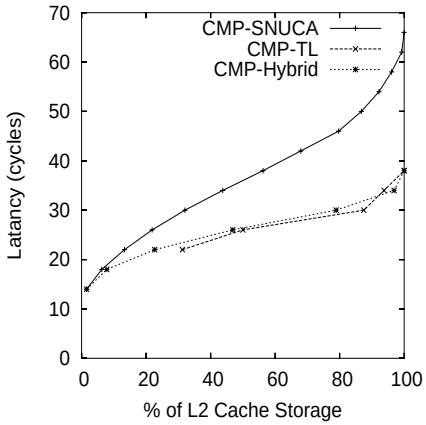

Figure 4. CMP-SNUCA vs. CMP-TLC vs. CMP-Hybrid Uncontended L2 Hit Latency

(8 bytes). The transmission line links of CMP-Hybrid provide low latency access to those blocks that tend to congregate in the center banks of the block migrating NUCA cache, Section 5.3.

Figure 4 compares the uncontended L2 cache hit latency between the CMP-SNUCA, CMP-TLC, and CMP-Hybrid designs. The plotted hit latency includes L1 miss latency, i.e. it plots the load-to-use latency for L2 hits. While CMP-TLC achieves a much lower average hit latency than CMP-SNUCA, CMP-SNUCA exhibits lower latency to the closest 1 MB to each processor. For instance, Figure 4 shows all processors in the CMP-SNUCA design can access their local bankcluster (6.25% of the entire cache) in 18 cycles or less. CMP-DNUCA attempts to maximize the hits to this closest 6.25% of the NUCA cache through migration, while CMP-TLC utilizes a much simpler logical design and provides fast access for all banks. CMP-Hybrid uses transmission lines to attain similar average hit latency as CMP-TLC, as well as achieving fast access to more banks than CMP-SNUCA.

3 Methodology

We evaluated all cache designs using full system simulation of a SPARC V9 CMP running Solaris 9. Specifically, we used Simics [33] extended with the out-of-order processor model, TFSim [34], and a memory system timing model. Our memory system implements a two-level directory cache-coherence protocol with sequential memory consistency. The intra-chip MSI coherence protocol maintains inclusion between the shared L2 cache and all on-chip L1 caches. All L1 requests and responses are sent via the L2 cache allowing the L2 cache to maintain up-to-date L1 sharer knowledge. The inter-chip MOSI coherence protocol maintains directory state at the off-chip memory controllers and only tracks which CMP nodes contain valid block copies. Our memory system timing model includes a detailed model of the intra- and inter-chip network. Our network models all messages communicated in the system including all requests, responses, replacements, and acknowledgements. Network routing is performed using a virtual cut-through scheme with infinite buffering at the switches.

Table 2. Evaluation Methodology

Bench	Fast Forward	Warm-up	Executed
Commercial Workloads (unit = transactions)			
apache	500000	2000	500
zeus	500000	2000	500
jbb	1000000	15000	2000
oltp	100000	300	100
Scientific Workloads (unit = billion instructions)			
barnes	None	1.9	run completion
ocean	None	2.4	run completion
apsi	88.8	4.64	loop completion
fma3d	190.4	2.08	loop completion

We studied the CMP cache designs for various commercial and scientific workloads. Alameldeen, et al. described in detail the four commercial workloads used in this study [2]. We also studied four scientific workloads: two Splash2 benchmarks [48]: barnes (16k-particles) and ocean (514×514), and two SPECOMP benchmarks [4]: apsi and fma3d. We used a work-related throughput metric to address multithreaded workload variability [2]. Thus for the commercial workloads, we measured transactions completed and for the scientific workloads, runs were completed after the cache warm-up period indicated in Table 2. However, for the specOMP workloads using the reference input sets, runs were too long to be completed in a reasonable amount of time. Instead, these loop-based benchmarks were split by main loop completion. This allowed us to evaluate all workloads using throughput metrics, rather than IPC.

4 Strided Prefetching

Both on and off-chip strided prefetching significantly improve the performance of our CMP-SNUCA baseline. Figure 5 presents runtime results for no prefetching, L2 prefetching only, and L1 and L2 prefetching combined, normalized to no prefetching. Error bars signify the 95% confidence intervals [3] and the absolute runtime (in 10K instructions per transaction/scientific benchmark) of the no prefetch case is presented below. Figure 5 illustrates the substantial benefit from L2 prefetching, particularly for regular scientific workloads. L2 prefetching reduces the run times of ocean and apsi by 43% and 59%, respectively. Strided L2 prefetching also improves performance of the commercial workloads by 4% to 17%.

The L1&L2 prefetching bars of Figure 5 indicate on-chip prefetching between each processor's L1 I and D caches and the shared L2 cache improves performance by an addi-

Table 3. Prefetching Characteristics

	L1 I Cache			L1 D Cache			L2 Cache		
benchmark	prefetches	coverage	accuracy	prefetches	coverage	accuracy	prefetches	coverage	accuracy
apache	7.6	18.3%	48.7%	5.3	9.3%	61.7%	7.0	39.0%	49.8%
jbb	2.5	30.1	57.0	2.0	7.7	35.9	3.0	38.3	36.8
oltp	15.1	26.7	53.5	1.4	5.9	59.6	1.9	35.0	50.3
zeus	11.3	19.0	47.9	4.9	15.7	78.6	8.0	47.2	56.7
barnes	0.0	9.1	38.7	0.2	3.0	20.8	0.0	12.3	22.8
ocean	0.0	22.1	50.9	17.6	85.9	88.3	4.0	91.3	87.5
apsi	0.0	10.8	38.5	5.6	46.7	99.4	5.7	98.7	98.8
fma3d	0.0	12.4	33.1	6.7	32.8	82.5	11.2	36.8	67.9

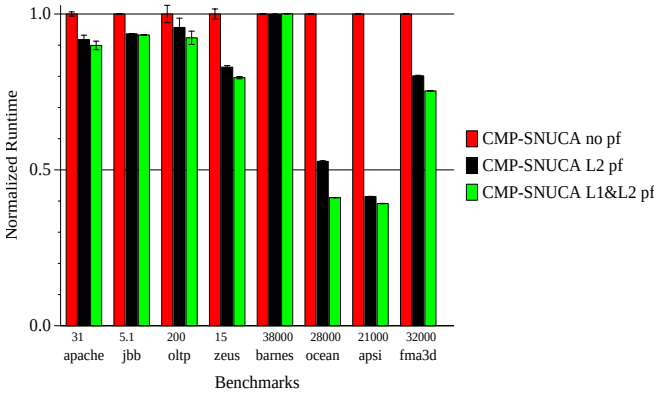

Figure 5. Normalized Execution: Strided Prefetching

tional 6% on average. On-chip prefetching benefits all benchmarks except for jbb and barnes, which have high local L1 cache hit rates of 91% and 99% respectively. Table 3 breaks down the performance of stride prefetching into:

$$prefetch\ rate = \frac{prefetches}{instructions/1000} \quad \text{(prefetches), and}$$

$$coverage\ \% = \frac{prefetchHits}{prefetchHits + misses} \times 100 \quad \text{(coverage), and}$$

$$accuracy\ \% = \frac{prefetchHits}{prefetches} \times 100 \quad \text{(accuracy).}$$

A Prefetch hit is defined as the first reference to a prefetched block including a "partial hit" to a block still in-flight. Except for L2 prefetches in jbb and barnes, less than 12% of prefetch hits were partial hits. Overall, as communication latency increases, the significant performance improvement attainable by prefetching ensures its continued integration into future high performance memory systems.

5 Block Migration

CMP caches utilizing block migration must effectively manage multiple processor working sets in order to reduce cache access latency. Although Kim, et al. [27] showed block migration significantly reduced cache access latency in a non-prefetching uniprocessor NUCA cache, most future large on-chip caches will implement hardware-directed prefetching and service multiple on-chip processors. Our CMP-DNUCA design extends block migration to an 8-processor CMP cache and supports strided prefetching at the L1 and L2 cache levels. We characterize the working sets existing in a shared CMP cache in Section 5.1. Then we describe our CMP cache implementing block migration in Section 5.2, and present evaluation results in Section 5.3.

5.1 Characterizing CMP Working Sets

We analyze the working sets of the workloads running on an eight-processor CMP. Specifically, we focus on understanding the potential benefits of block migration in a CMP cache by observing the behavior of L2 cache blocks. We model a single-banked 16 MB inclusive shared L2 cache with 16-way associativity and a uniform access time to isolate the sharing activity from access latency and conflict misses. No prefetching is performed in these runs. To mitigate cold start effects, the runs are long enough that L2 cache misses outnumber physical L2 cache blocks by an order of magnitude.

Figure 6 shows the cumulative distribution of the number of processors that access each block. For the scientific workloads, the vast majority of all blocks—between 70.3% and 99.9%—are accessed by a single processor. Somewhat surprisingly, even the commercial workloads share relatively few blocks. Only 5% to 28% of blocks in the evaluated commercial workloads are accessed by more than two processors. Because relatively few blocks are shared across the entire workload spectrum, block migration can potentially improve all workloads by moving blocks towards the single processor that requests them.

Although relatively few blocks are shared in all workloads, a disproportionate fraction of L2 hits are satisfied by highly shared blocks for the commercial workloads. Figure 7 shows the cumulative distribution of L2 hits for blocks accessed by 1 to 8 processors. The three array-based scientific workloads (fma3d [4], apsi [4], and ocean [48]) exhibit extremely low inter-processor request sharing. Less than 9%

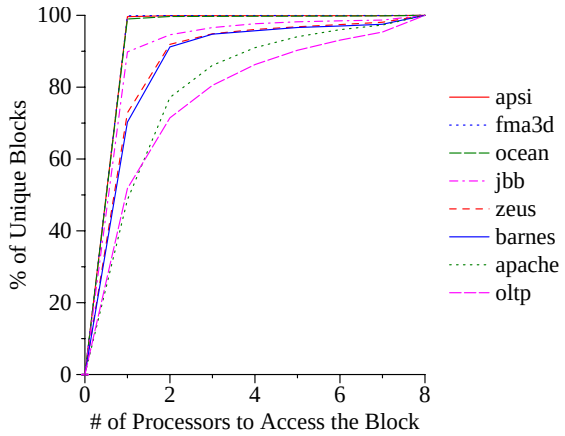

Figure 6. Cumulative Percentage of Unique L2 Blocks vs. # of Processors to Access the Block During its L2 Cache Lifetime

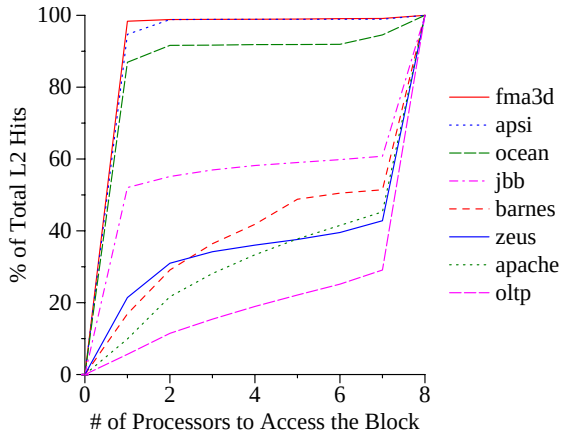

Figure 7. Cumulative Percentage of Total L2 Cache Hits vs. # of Processors to Access a Block During its L2 Cache Lifetime

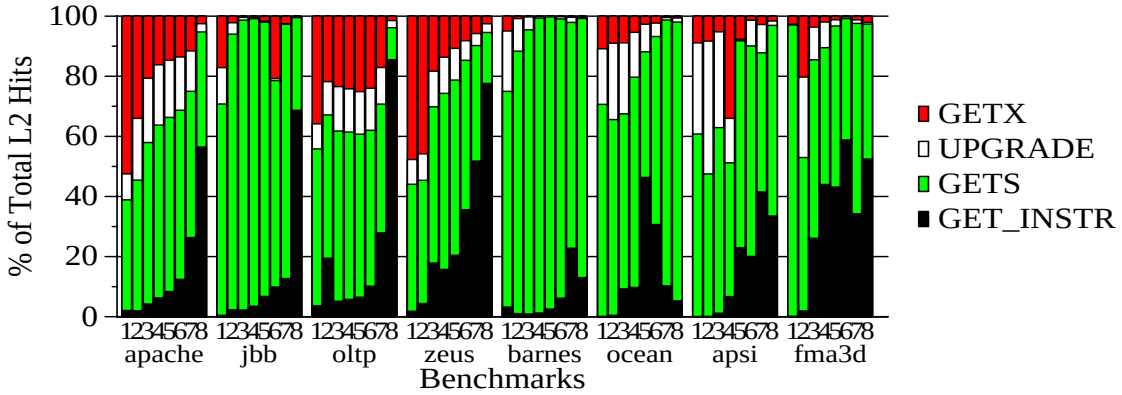

Figure 8. Request Type Distribution vs. # of Processors to Access a Block During its L2 Cache Lifetime

of all L2 hits are to blocks shared by multiple processors. However, for barnes, utilizing a tree data structure [48], 71% of L2 hits are to blocks shared among multiple processors. For the commercial workloads, more than 39% of L2 hits are to blocks shared by *all* processors, with as many as 71% of hits for oltp. Figure 8 breaks down the request type over the number of processors to access a block. In the four commercial workloads, instruction requests (GET_INSTR) make up over 56% of the L2 hits to blocks shared by all processors. Kundu et al. [31] recently confirmed the high degree of instruction sharing in a shared CMP cache running oltp. The large fraction of L2 hits to highly shared blocks complicates block migration in CMP-DNUCA. Since shared blocks will be pulled in all directions, these blocks tend to congregate in the center of the cache rather than toward just one processor.

5.2 Implementing CMP Block Migration

Our CMP-DNUCA implementation employs block migration within a shared L2 cache. This design strives to reduce additional state while providing correct and efficient allocation, migration and search policies.

Allocation. The allocation policy seeks an efficient initial placement for a cache block, without creating excessive cache conflicts. While 16-way set-associative banks help reduce conflicts, the interaction between migration and allocation can still cause unnecessary replacements. CMP-DNUCA implements a simple, static allocation policy that uses the low-order bits of the cache tag to select a bank within the block's bankset (i.e., the bankcluster). This simple scheme works well across most workload types. While not studied in this paper, we conjecture that static allocation also works well for heterogeneous workloads, because all active processors will utilize the entire L2 cache storage.

Migration. We investigated several different migration policies for CMP-DNUCA. A migration policy should maximize the proportion of L2 hits satisfied by the banks closest to a processor. Directly migrating blocks to a requesting processor's local bankcluster increases the number of local bankcluster hits. However, direct migration also increases the proportion of costly remote hits satisfied by a distant processors' local bankcluster. Instead CMP-DNUCA implements a gradual migration policy that moves blocks along the six bankcluster chain:

$$\begin{matrix} other \\ local \end{matrix} \Rightarrow \begin{matrix} other \\ inter \end{matrix} \Rightarrow \begin{matrix} other \\ center \end{matrix} \Rightarrow \begin{matrix} my \\ center \end{matrix} \Rightarrow \begin{matrix} my \\ inter \end{matrix} \Rightarrow \begin{matrix} my \\ local \end{matrix}$$

The gradual migration policy allows blocks frequently accessed by one processor to congregate near that particular processor, while blocks accessed by many processors tend to move within the center banks. Furthermore the policy separates the different block types without requiring extra state or complicated decision making. Only the current bank location and the requesting processor id is needed to determine which bank, if any, a block should be moved to.

Search. Similar to the best performing uniprocessor D-NUCA search policy, we advocate using a two-phase multicast search for CMP-DNUCA. The goal of the two-phase search policy is to maximize the number of first phase hits, while limiting the number of futile request messages. Based on the previously discussed gradual migration policy, hits most likely occur in one of six bankclusters: the requesting processor's local or inter bankclusters, or the four center bankclusters. Therefore the first phase of our search policy broadcasts a request to the appropriate banks within these six bankclusters. If all six initial bank requests miss, we broadcast the request to the remaining 10 banks of the bankset. Only after a request misses in all 16 banks of the bankset will a request be sent off chip. Waiting for 16 replies over two phases adds significant latency to cache misses. Unfortunately, as discussed in Section 2.3, implementing a smart search mechanism to minimize search delay in CMP-DNUCA is difficult. Instead, we provide results in the following section for an idealized smart search mechanism.

A unique problem of CMP-DNUCA is the potential for *false misses*, where L2 requests fail to find a cache block because it is in transit from one bank to another. It is essential that false misses are not naively serviced by the directory, otherwise two valid block copies could exist within the L2 cache creating a coherence nightmare. One possible solution is requests could consult a second set of centralized on-chip tags not effected by block migration before going off chip. However, this second tag array would cost over 1 MB of extra storage and require approximately 1 KB of data to be read and compared on every lookup—because each set logically appears 256-way associative.

Instead, CMP-DNUCA compensates for false misses by relying on the directory sharing state to indicate when a possible false miss occurred. If the directory believes a valid block copy already exists on chip, the L2 cache stops migrations and searches for an existing valid copy before using the received data. Only after sequentially examining all banks of a bankset with migration disabled will a cache be certain the block isn't already allocated. While the meticulous examination ensures correctness, it is very slow. Therefore, it is important to ensure false misses don't happen frequently.

We significantly reduced the frequency of false misses by implementing a lazy migration mechanism. We observed

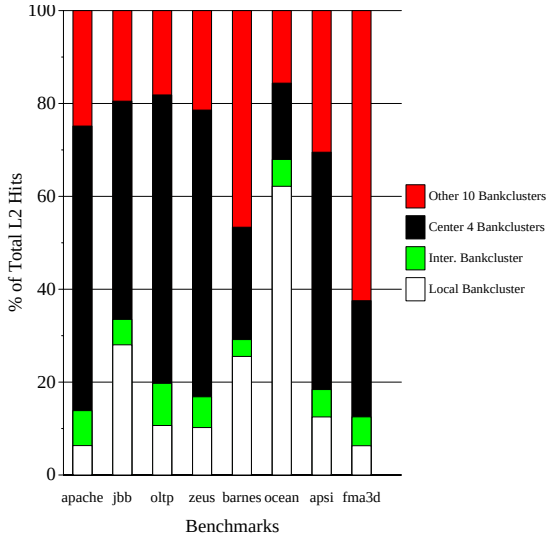

Figure 9. L2 Hit Distribution of CMP-DNUCA

that almost all false misses occur for a few hot blocks that are rapidly accessed by multiple processors. By delaying migrations by a thousand cycles, and canceling migrations when a different processor accesses the same block, CMP-DNUCA still performs at least 94% of all scheduled migrations, while reducing false misses by at least 99%. In apsi, for instance, lazy migration reduced the fraction of false misses from 18% of all misses to less than 0.00001%.

5.3 Evaluating CMP Block Migration

Our CMP-DNUCA evaluation shows block migration creates a performance degradation unless a smart search mechanism is utilized. A smart search mechanism reduces contention by decreasing the number of unsuccessful request messages and reduces L2 miss latency by sending L2 misses directly off chip before consulting all 16 banks of a bankset.

The high demand for the equally distant central banks restricts the benefit of block migration for the commercial workloads. Figure 9 shows in all four commercial workloads over 47% of L2 hits are satisfied in the center bankclusters. The high number of central hits directly correlates to the increased sharing in the commercial workloads previously shown in Figure 7. Figure 10 graphically illustrates the distribution of L2 hits for oltp, where the dark squares in the middle represent the heavily utilized central banks.

Conversely, CMP-DNUCA exhibits a mixture of behavior running the four scientific workloads. Due to a lack of frequently repeatable requests, barnes, apsi, and fma3d encounter 30% to 62% of L2 hits in the distant 10 bankclusters. These hits cost significant performance because the distant banks are only searched during the second phase. On the other hand, the scientific workload ocean contains repeatable requests and exhibits very little sharing. Figure 9 indicates CMP-DNUCA successfully migrates over 60% of ocean's L2 hits to the local bankclusters. The dark colored squares of

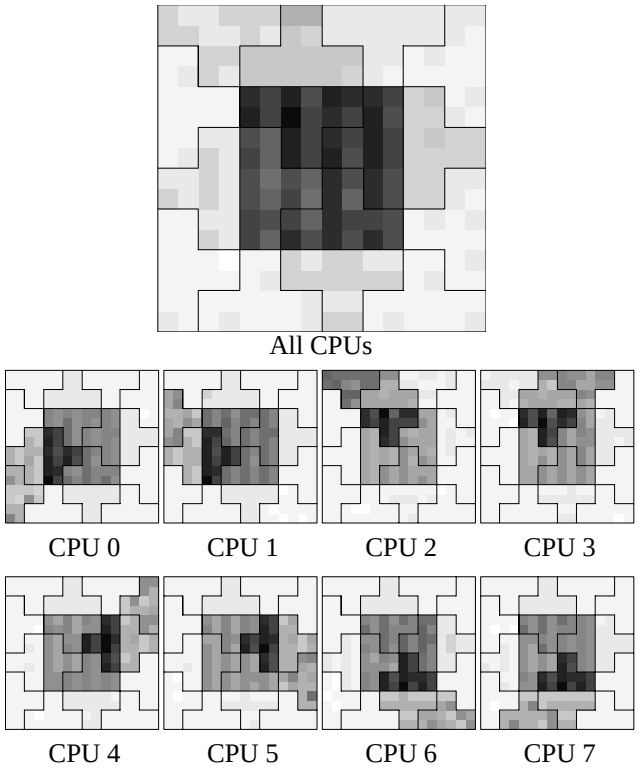

Figure 10. oltp L2 Hit Distribution

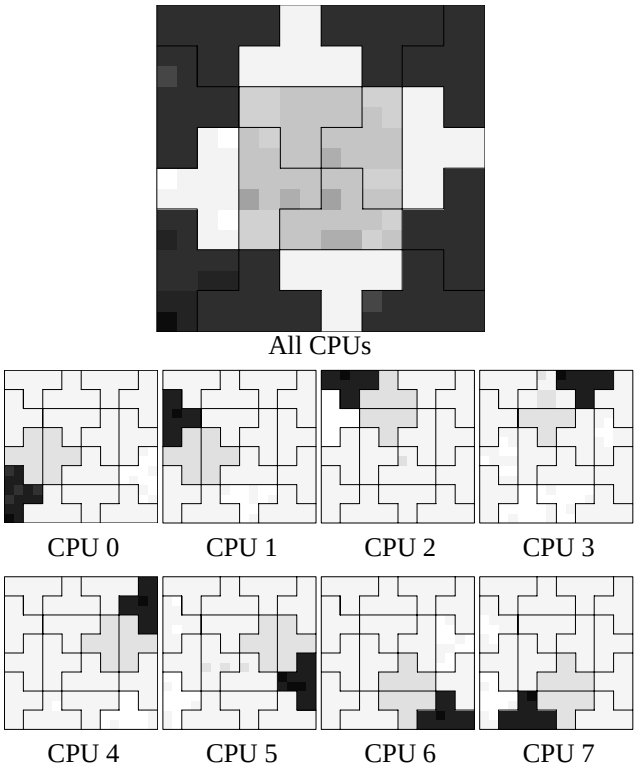

Figure 11. ocean L2 Hit Distribution

The figures above illustrate the distribution of cache hits across the L2 cache banks. The Tetris shapes indicate the bankclusters and the shaded squares represent the individual banks. The shading indicates the fraction of all L2 hits to be satisfied by a given bank, with darker being greater. The top figure illustrates all hits, while the 8 smaller figures illustrate each CPU's hits.

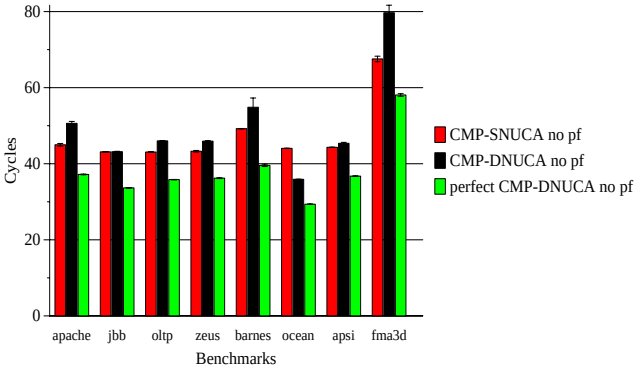

Figure 12. Avg. L2 Hit Latency: No Prefetching

Figure 11 graphically display how well CMP-DNUCA is able to split ocean's data set into the local bankclusters.

The limited success of block migration along with the slow two-phase search policy causes CMP-DNUCA to actually increase L2 hit and L2 miss latency. Figure 12 indicates CMP-DNUCA only reduces L2 hit latency versus CMP-SNUCA for one workload, ocean. The latency increase for the other seven workloads results from second phase hits encountering 31 to 51 more delay cycles than CMP-SNUCA L2 hits. The added latency of second phase hits in CMP-DNUCA is due to the delay waiting for responses from the first phase requests. Furthermore due to the slow two-phase search policy, L2 misses also encounter 23 to 65 more delay cycles compared to CMP-SNUCA.

A smart search mechanism solves CMP-DNUCA's slow search problems. Figure 12 shows the L2 hit latency attained by CMP-DNUCA with perfect search (perfect CMP-DNUCA), where a processor sends a request directly to the cache bank storing the block. Perfect CMP-DNUCA reduces L2 hit latency across all workloads by 7 to 15 cycles versus CMP-SNUCA. Furthermore, when the block isn't on chip, perfect CMP-DNUCA immediately generates an off-chip request, allowing its L2 miss latency to match that of CMP-SNUCA. Although the perfect search mechanism is infeasible, architects may develop practical smart search schemes in the future. We assume perfect searches for the rest of the paper to examine the potential benefits of block migration.

6 Targeting On-chip Latency In Isolation

On-chip prefetching performs competitively with the more extravagant techniques of block migration and transmission lines. A comparison between the bars labeled CMP-SNUCA L1&L2 pf to those labeled perfect CMP-DNUCA

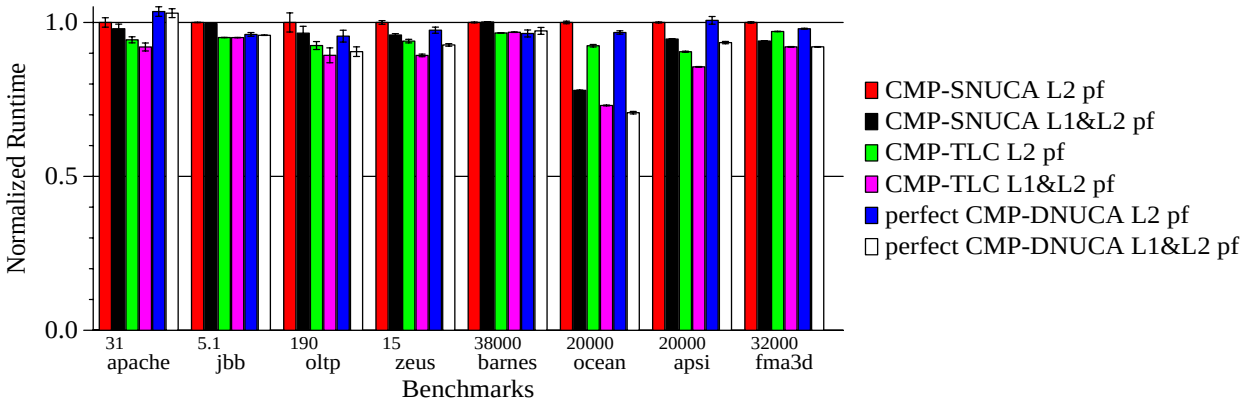

Figure 13. Normalized Execution: Latency Reduction Techniques with Prefetching

L2 pf and CMP-TLC L2 pf in Figure 13 reveals on-chip prefetching achieves the greatest single workload improvement over the baseline (CMP-SNUCA L2 pf)—22% for ocean. In addition, on-chip prefetching improves performance by at least 4% for the workloads zeus, apsi, and fma3d.

Block migration improves performance by 2%–4% for 6 of the 8 workloads. However, for apache, an increase in cache conflicts causes a 4% performance loss. As illustrated by Figure 9, the four center bankclusters (25% of the total L2 storage) incur 60% of the L2 hits. The unbalanced load increases cache conflicts, resulting in a 13% increase in L2 misses versus the baseline design.

By directly reducing wire latency, transmission lines consistently improve performance, but bandwidth contention prevents them from achieving their full potential. Figure 13 shows transmission lines consistently improve performance between 3% to 10% across all workloads—8% on average. However, CMP-TLC would do even better, except for bandwidth contention that accounts for 26% of the L2 hit latency.

Overall, CMP-TLC is likely to improve a larger number of workloads because transmission lines reduce latency without relying on predictable workload behavior. On the other hand, prefetching and block migration potentially provide a greater performance improvement for a smaller number of workloads.

7 Merging Latency Management Techniques

None of the three evaluated techniques subsume another, but rather the techniques can work in concert to manage wire latency. Section 7.1 demonstrates prefetching is mostly orthogonal to transmission lines and block migration, while Section 7.2 evaluates combining all three techniques: prefetching, block migration, and transmission lines.

7.1 Combining with On-chip Prefetching

Prefetching, which reduces latency by predicting the next cache block to be accessed, is orthogonal to block migration. However, for some scientific workloads, prefetching's benefit slightly overlaps the consistent latency reduction of CMP-TLC. The bars labeled CMP-DNUCA L1&L2 pf and CMP-TLC L1&L2 pf in Figure 13 show the performance of CMP-DNUCA and CMP-TLC combined with on-chip prefetching. A comparison between the L2 pf bars and the L1&L2 pf bars reveals L1 prefetching provides roughly equal improvement across all three designs. The only slight deviation is on-chip prefetching improves CMP-TLC by 5% to 21% for the scientific workloads ocean, apsi, and fma3d, while improving CMP-DNUCA by 6% to 27%. While combining each technique with prefetching is straightforward, combining all three techniques together requires a different cache design.

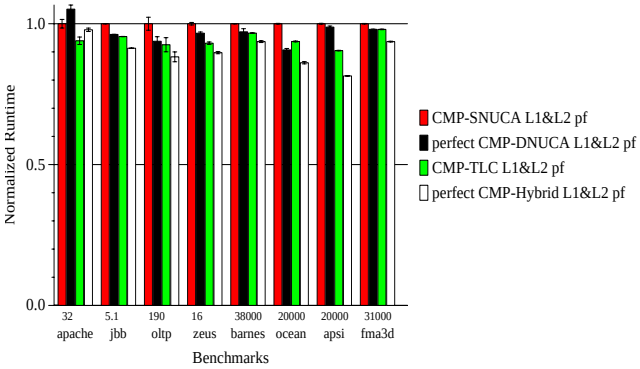

Figure 14. Normalized Execution: Combining All Techniques

7.2 Combining All Techniques

CMP-Hybrid combines prefetching and transmission lines and block migration to achieve the best overall performance. Figure 14 shows that CMP-Hybrid combined with on and off-chip prefetching (perfect CMP-Hybrid L1&L2 pf) reduces runtime by 2% to 19% compared to the baseline design. As previously shown in Figure 9, 25% to 62% of L2 hits in CMP-DNUCA are to the center banks for all workloads except ocean. By providing low-latency access to the center banks, Figure 15 indicates CMP-Hybrid (bars labelled H) reduces the average L2 hit latency for these seven work-

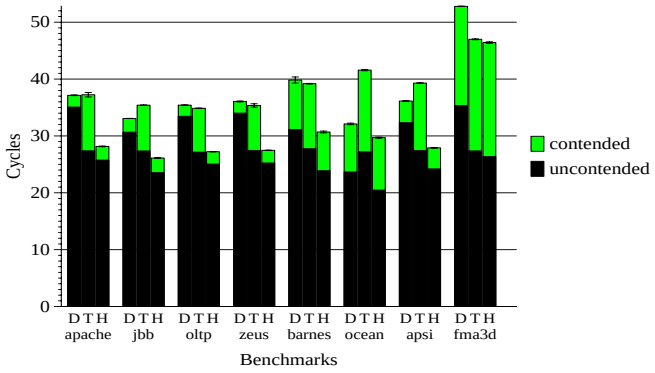

Figure 15. Avg. L2 Hit Latency: Combining Latency Reduction Techniques

loads by 6 to 9 cycles versus CMP-DNUCA (bars labelled D). As previously mentioned in Section 6, link contention within the centralized network of CMP-TLC accounts for 26% of delay for L2 hits. CMP-Hybrid's high bandwidth, distributed network reduces contention, allowing for better utilization of transmission line's low latency. Figure 15 shows L2 hits within CMP-Hybrid encounter up to 8 fewer contention delay cycles as compared those within CMP-TLC (bars labelled T).

While CMP-Hybrid achieves impressive performance, one should note it also relies on a good search mechanism for its performance. Furthermore, CMP-Hybrid requires both extra manufacturing cost to produce on-chip transmission lines and additional verification effort to implement block migration.

8 Energy Efficiency

Although the majority of L2 cache power consumption is expected to be leakage power [14], we analyze each network's dynamic energy-delay product to determine the efficiency of each design. Similar to our previous study [6], we estimate the network energy by measuring the energy used by the wires as well as the switches. For conventional RC interconnect using repeaters, we measure the energy required to charge and discharge the capacitance of each wire segment. For transmission lines, we measure the energy required to create the incident wave. We do not include the dynamic energy consumed within the L2 cache banks, but we do note block migration requires accessing the storage banks about twice as often as the static designs.

Prefetching and block migration improve network efficiency for some scientific workloads, while transmission lines potentially improve efficiency across all workloads. Figure 16 plots the product of the networks' dynamic energy consumption and the design's runtime normalized to the value of the CMP-SNUCA design running ocean. The two block migrating designs, CMP-DNUCA and CMP-Hybrid, assume the perfect search mechanism. Figure 16 shows the high accuracy and coverage of L1 and L2 prefetching results

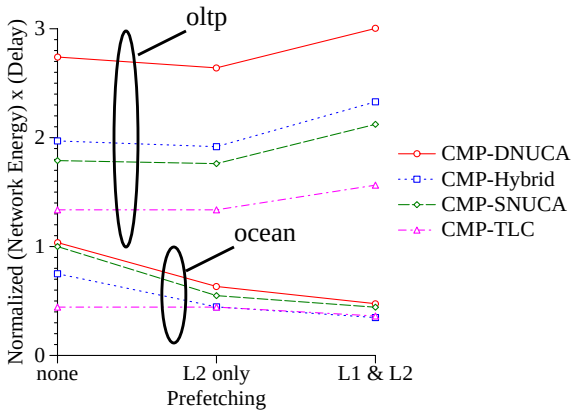

Figure 16. Normalized Network Energy-Delay: oltp and ocean

in a reduction of network energy-delay by 18% to 54% for all designs with ocean. Also, by successfully migrating frequently requested blocks to the more efficiently accessed local banks, CMP-DNUCA achieves similar network efficiency as CMP-SNUCA for ocean despite sending extra migration messages. However, both block migration and L1 prefetching increase network energy-delay between 17% and 53% for the commercial workload oltp. On the other hand, those designs using transmission lines, CMP-TLC and CMP-Hybrid, reduce network energy-delay by 26% for oltp and 33% for ocean on average versus their counterpart designs that exclusively rely on conventional wires, CMP-SNUCA and CMP-DNUCA. In general, workload characteristics affect prefetching and block migration efficiency more than transmission line efficiency.

9 Conclusions

Managing on-chip wire delay in CMP caches is essential in order to improve future system performance. Strided prefetching is a common technique utilized by current designs to tolerate wire delay. As wire delays continue to increase, architects will turn to additional techniques such as block migration or transmission lines to manage on-chip delay. While block migration effectively reduces wire delay in uniprocessor caches, we discover block migration's capability to improve CMP performance relies on a difficult to implement smart search mechanism. Furthermore, the potential benefit of block migration in a CMP cache is fundamentally limited by the large amount of inter-processor sharing that exists in some workloads. On the other hand, on-chip transmission lines consistently improve performance, but their limited bandwidth becomes a bottleneck when combined with on-chip prefetching. Finally, we investigate a hybrid design, which merges transmission lines, block migration, and prefetching. Adding transmission lines from each processor to the center of the NUCA cache could alleviate the deficiencies of implementing block migration or transmission lines alone.

Acknowledgements

We thank Alaa Alameldeen, Mark Hill, Mike Marty, Kevin Moore, Phillip Wells, Allison Holloway, Luke Yen, the Wisconsin Computer Architecture Affiliates, Virtutech AB, the Wisconsin Condor group, the Wisconsin Computer Systems Lab, and the anonymous reviewers for their comments on this work.

References

[1] V. Agarwal, S. W. Keckler, and D. Burger. The Effect of Technology Scaling on Microarchitectural Structures. *Technical Report TR-00-02, Department of Computer Sciences, UT at Austin*, May 2001.

[2] A. R. Alameldeen, M. M. K. Martin, C. J. Mauer, K. E. Moore, M. Xu, D. J. Sorin, M. D. Hill, and D. A. Wood. Simulating a $2M Commercial Server on a $2K PC. *IEEE Computer*, 36(2):50–57, Feb. 2003.

[3] A. R. Alameldeen and D. A. Wood. Variability in Architectural Simulations of Multi-threaded Workloads. In *Proceedings of the Ninth IEEE Symposium on High-Performance Computer Architecture*, pages 7–18, Feb. 2003.

[4] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W. Jones, and B. Parady. SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance. In *Workshop on OpenMP Applications and Tools*, pages 1–10, July 2001.

[5] 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 *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 282–293, June 2000.

[6] B. M. Beckmann and D. A. Wood. TLC: Transmission Line Caches. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2003.

[7] R. Chandra, S. Devine, B. Verghese, A. Gupta, and M. Rosenblum. Scheduling and Page Migration for Multiprocessor Compute Servers. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 1994.

[8] R. T. Chang, N. Talwalkar, C. P. Yue, and S. S. Wong. Near Speed-of-Light Signaling Over On-Chip Electrical Interconnects. *IEEE Journal of Solid-State Circuits*, 38(5):834–838, May 2003.

[9] T.-F. Chen and J.-L. Baer. Reducing Memory Latency via Non-Blocking and Prefetching Caches. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 51–61, Oct. 1992.

[10] T.-F. Chen and J.-L. Baer. A Performance Study of Software and Hardware Data Prefetching Schemes. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 223–232, Apr. 1994.

[11] T.-F. Chen and J.-L. Baer. Effective Hardware-Based Data Prefetching for High Performance Processors. *IEEE Transactions on Computers*, 44(5):609–623, May 1995.

[12] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2003.

[13] F. Dahlgren and P. Stenström. Effectiveness of Hardware-Based Stride and Sequential Prefetching in Shared-Memory Multiprocessors. In *Proceedings of the First IEEE Symposium on High-Performance Computer Architecture*, pages 68–77, Feb. 1995.

[14] B. Doyle, R. Arghavani, D. Barlage, S. Datta, M. Doczy, J. Kavalieros, A. Murthy, and R. Chau. Transistor Elements for 30nm Physical Gate Lengths and Beyond. *Intel Technology Journal*, May 2002.

[15] B. Falsafi and D. A. Wood. Reactive NUMA: A Design for Unifying S-COMA and CC-NUMA. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 229–240, June 1997.

[16] I. T. R. for Semiconductors. ITRS 2003 Edition. Semiconductor Industry Association, 2003. http://public.itrs.net/Files/2003ITRS/Home2003.htm.

[17] E. Hagersten, A. Landin, and S. Haridi. DDM–A Cache-Only Memory Architecture. *IEEE Computer*, 25(9):44–54, Sept. 1992.

[18] L. Hammond, B. Hubbert, M. Siu, M. Prabhu, M. Chen, and K. Olukotun. The Stanford Hydra CMP. *IEEE Micro*, 20(2):71–84, March-April 2000.

[19] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. The microarchitecture of the Pentium 4 processor. *Intel Technology Journal*, Feb. 2001.

[20] R. Ho, K. W. Mai, and M. A. Horowitz. The Future of Wires. *Proceedings of the IEEE*, 89(4):490–504, Apr. 2001.

[21] T. Horel and G. Lauterbach. UltraSPARC-III: Designing Third Generation 64-Bit Performance. *IEEE Micro*, 19(3):73–85, May/June 1999.

[22] D. Joseph and D. Grunwald. Prefetching Using Markov Predictors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 252–263, June 1997.

[23] 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, May 1990.

[24] R. Kalla, B. Sinharoy, and J. M. Tendler. IBM Power5 Chip: A Dual Core Multithreaded Processor. *IEEE Micro*, 24(2):40–47, Mar/Apr 2004.

[25] M. Karlsson, K. E. Moore, E. Hagersten, and D. A. Wood. Memory System Behavior of Java-Based Middleware. In *Proceedings of the Ninth IEEE Symposium on High-Performance Computer Architecture*, pages 217–228, Feb. 2003.

[26] R. E. Kessler, R. Jooss, A. Lebeck, and M. D. Hill. Inexpensive Implementations of Set-Associativity. *16th Annual International Symposium on Computer Architecture*, May 1989.

[27] C. Kim, D. Burger, and S. W. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches. *10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2002.

[28] P. Kongetira. A 32-way Multithreaded SPARCÆ Processor. In *Proceedings of the 16th HotChips Symposium*, Aug. 2004.

[29] G. K. Konstadinidis and et. al. Implementation of a Third-Generation 1.1-GHz 64-bit Microprocessor. *IEEE Journal of Solid-State Circuits*, 37(11):1461–1469, Nov 2002.

[30] K. Krewell. UltraSPARC IV Mirrors Predecessor. *Microprocessor Report*, pages 1–3, Nov. 2003.

[31] P. Kundu, M. Annavaram, T. Diep, and J. Shen. A Case for Shared Instruction Cache on Chip Multiprocessors running OLTP. *Computer Architecture News*, 32(3):11–18, 2004.

[32] C. Liu, A. Sivasubramaniam, and M. Kandemir. Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs. In *Proceedings of the Tenth IEEE Symposium on High-Performance Computer Architecture*, Feb. 2004.

[33] P. S. Magnusson et al. Simics: A Full System Simulation Platform. *IEEE Computer*, 35(2):50–58, Feb. 2002.

[34] C. J. Mauer, M. D. Hill, and D. A. Wood. Full System Timing-First Simulation. *2002 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 108–116, June 2002.

[35] C. McNairy and D. Soltis. Itanium 2 Processor Microarchitecture. *IEEE Micro*, 23(2):44–55, March/April 2003.

[36] H. E. Mizrahi, J.-L. Baer, E. D. Lazowska, and J. Zahorjan. Introducing Memory into the Switch Elements of Multiprocessor Interconnection Networks. In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, May 1989.

[37] B. A. Nayfeh and K. Olukotun. Exploring the Design Space for a Shared-Cache Multiprocessor. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, Apr. 1994.

[38] A. Pajuelo, A. Gonz·lez, and M. Valero. Speculative Dynamic Vectorization. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, May 2002.

[39] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides. Design Tradeoffs for the Alpha EV8 Conditional Branch Predictor*. *29th Annual International Symposium on Computer Architecture*, May 2002.

[40] K. So and R. N. Rechtschaffen. Cache Operations by MRU Change. *IEEE Transactions on Computers*, 37(6):700–709, June 1988.

[41] V. Soundararajan, M. Heinrich, B. Verghese, K. Gharachorloo, A. Gupta, and J. Hennessy. Flexible Use of Memory for Replication/Migration in Cache-Coherent DSM Multiprocessors. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 342–355, June 1998.

[42] D. Sylvester and K. Keutzer. Getting to the Bottom of Deep Submicron II: a Global Wiring Paradigm. In *Proceedings of the 1999 International Symposium on Physical Design*, pages 193–200, 1999.

[43] J. M. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy. POWER4 System Microarchitecture. IBM Server Group Whitepaper, Oct. 2001.

[44] J. M. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy. POWER4 System Microarchitecture. *IBM Journal of Research and Development*, 46(1), 2002.

[45] M. Tremblay, J. Chan, S. Chaudhry, A. W. Conigliaro, and S. S. Tse. The MAJC Architecture: A Synthesis of Parallelism and Scalability. *IEEE Micro*, 20(6):12–25, November-December 2000.

[46] D. M. Tullsen and S. J. Eggers. Limitations of Cache Prefetching on a Bus-Based Multiprocessor. *20th Annual International Symposium on Computer Architecture*, pages 278–288, May 1993.

[47] J. Turley. *The Essential Guide to Semiconductors*. Prentice Hall, 2003.

[48] 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 Annual International Symposium on Computer Architecture*, pages 24–37, June 1995.

[49] Z. Zhang and J. Torrellas. Speeding up Irregular Applications in Shared-Memory Multiprocessors: Memory Binding and Group Prefetching. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 188–199, June 1995.