Computer Sciences Department

Balancing Capacity and Latency in CMP Caches

Bradford Beckmann
Michael Marty
David Wood

Technical Report #1554

February 2006

UNIVERSITY OF WISCONSIN MADISON

Balancing Capacity and Latency in CMP Caches

Bradford M. Beckmann, Michael R. Marty, and David A. Wood

Computer Sciences Department
University of Wisconsin-Madison
{beckmann, mikem, markhill, david}@cs.wisc.edu

Abstract

The large working sets of commercial and scientific workloads stress the L2 caches of Chip Multiprocessors (CMPs). Some CMPs use a shared L2 cache, to maximize the on-chip cache capacity and minimize misses. Others use private L2 caches, replicating data to limit the delay due to global wires and minimize cache access time. Recent hybrid proposals strive to balance latency and capacity, but use simple, static rules that are not robust to changes in workload behavior and system configuration.

This paper studies alternative L2 cache designs for an 8-processor CMP system and shows that two previous selective-replication mechanisms actually degrade performance up to 13%, for some combinations of scientific and commercial workloads and system configurations. We propose the Adaptive Selective Replication (ASR) mechanism that dynamically monitors workload behavior to control replication. ASR replicates read-only blocks only when it estimates the benefit of replication (lower L2 hit latency) exceeds the cost (more L2 misses). Full-system simulation results show that ASR provides robust performance: decreasing runtimes by as much as 31% versus shared caches, 33% versus private caches, and 27% versus CMP-NuRapid and Victim Replication. Furthermore, while ASR does not improve the performance of all workloads on all system configurations, it almost always performs as well as the best alternative.

1 Introduction

As Chip Multiprocessors (CMPs) emerge in mainstream systems they must provide good performance for a wide variety of workloads across a range of system configurations. Level-2 (L2) cache management presents a key challenge, especially in the face of the conflicting requirements of reducing off-chip misses (capacity) and managing slow global wires (latency). Current CMP systems, such as the IBM Power 4/5 [8] and Sun Niagara [17], employ shared L2 caches to maximize the on-chip cache capacity by preventing replication. While shared caches minimize off-chip misses, they have higher access latencies since many requests must cross global wires to reach distant L2 banks. In contrast, private L2 caches [18, 22] reduce average access latency by replicating data close to the requesting processor, but sacrifice on-chip capacity and incur more misses.

Recent hybrid cache designs seek to achieve a balance between latency and capacity by selectively replicating cache blocks. CMP-NuRapid [6] and Victim Replication [33] have nominally private L2 caches, but replicate data blocks under certain fixed criteria. These schemes perform better than private and shared caches for selected workloads and system configurations. However, their static replication policies cannot adjust to match different workload and data set behavior.

This paper proposes an adaptive CMP cache organization that offers the latency advantage of private caches yet conserves space like a shared cache. The *Adaptive Selective Replication (ASR)* mechanism dynamically monitors cache behavior to determine how much replication of read-only data should exist between the L2

caches. The paper shows shared read-only data exhibits high request locality, and ASR exploits this behavior by selectively replicating frequently requested blocks when on-chip storage is under high demand. ASR manages the cache hierarchy on a per-processor basis, allowing cache capacity sharing between processors operating on the same data and cache isolation for processors operating on different data.

ASR's adaptive replication policy provides robust performance across a wide variety of workloads and system configurations. Thus, the same CMP will provide good performance in single-chip and multiple-chip CMP systems, with slower memory access times [14], and for scientific and commercial workloads.

This paper makes the following contributions:

- Shared read-only data account for up to 72% of L2 requests, but on average consume—without replication—only 5% of the L2 cache capacity. Furthermore in commercial workloads, requests for shared read-only data have tremendous locality: the top 3% of shared read-only blocks account for 70% of requests.
- The Adaptive Selective Replication (ASR) mechanism dynamically estimates the cost (extra misses) and benefit (lower hit latency) of replication, increasing or decreasing the replication level to minimize average access time. ASR adds only 1.2% storage overhead to the on-chip cache hierarchy, but reduces runtimes up to 27% versus both the Victim Replication [33] and CMP-NuRapid's replication [6] policy. ASR reduces runtimes by 11% and 12% on average versus the shared and private cache designs respectively, and by 5% on average versus the previous replication policies

Section 2 characterizes the CMP working sets of the 8 evaluated workloads, Section 3 introduces the baseline shared and private cache designs, and Section 4 describes ASR. Section 5 describes the evaluation methodology, Section 6 presents the results, and Section 7 summarizes related work.

2 Characterizing CMP Working Sets

This section focuses on understanding the potential benefits and drawbacks of replication in a CMP cache. This study simulates an eight-processor CMP executing various commercial and scientific workloads under the Solaris 9 operating system. Section 5 describes the simulation environment and workloads.

The analysis focuses on the behavior of cache blocks during their on-chip lifetime; that is, the interval from when a miss brings a block on chip until it is replaced. The simulation model assumes 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 most conflict misses. To mitigate cold start effects, all simulations run long enough that total L2 cache misses significantly outnumber the L2 block frames.

2.1 Sharing Types: Requests vs. Capacity

The cost and benefit of replication depends on the cache block's sharing behavior. We identify three distinct sharing types: 1. *Single Requestor* blocks are accessed by a single processor, 2. *Shared Read-Only* blocks are read, but not written, by multiple processors, and 3. *Shared Read-Write* blocks are accessed by multiple pro-

Table 1. L2 Cache Capacity Profile

Benchmark	Single Requestor		Shared Read-Only			Shared Read-Write		
	% of Requests	% of Capacity	% of Requests	% of Capacity	Avg. # of Sharers	% of Requests	% of Capacity	Avg. # of Sharers
apache	11%	55%	48%	17%	3.7	41%	29%	3.0
jbb	55	91	44	10	3.5	1	< 1	2.4
oltp	4	52	72	20	4.3	24	28	3.8
zeus	17	76	59	9	3.0	24	15	2.3
apsi	99	>99	< 1	< 1	7.3	< 1	< 1	2.8
art	49	62	51	38	3.0	< 1	< 1	2.7
galgel	< 1	84	< 1	< 1	4.0	99	16	5.3
mgrid	96	98	4	2	2.3	< 1	< 1	2.2

cessors, with at least one write. Single-requestor blocks cannot benefit from replication. Shared read-only and share read-write blocks can, but the latter will incur extra delay on writes due to coherence invalidations.

The percentage of requests to each sharing type varies significantly between the workloads. Table 1 shows that shared read-only requests dominate the four commercial workloads, 44-72% of requests, while Art, with 51%, is the only scientific workload to make more than 4% of its requests to shared read-only blocks. Single-requestor blocks account for nearly all the requests by Apsi and Mgrid, nearly half for Jbb and Art, and relatively little for the rest. Three of the four commercial workloads have many requests, 24-41%, to shared read-write blocks, but Galgel is the only scientific workload, at 99%, to have more than 1%.

While many requests are to shared data, single-requestor blocks consume the majority of the cache capacity. Table 1 shows that single-requestor blocks account for over 50% of L2 cache capacity for all workloads and over 90% Jbb, Apsi, and Mgrid. In comparison, shared read-only and shared read-write data consume relatively little capacity, with the maximum being less than 40%.

Replicating shared blocks in private caches to reduce access latency is attractive, since they are accessed frequently yet consume relatively little cache capacity. However, blind replication is dangerous, since the degree of sharing suggests that the capacity could increase significantly. Table 1 shows that the frequently requested shared read-only blocks in Apache, Jbb, Oltp, Zeus, and Art are requested by 3.0 to 4.3 processors during their on-chip cache lifetime. Fully replicating these blocks could increase the effective working set by 20-100%.

Fortunately, shared read-only blocks exhibit strong locality, especially for commercial workloads. Figure 1a plots the cumulative request distribution versus the cumulative capacity (block) distribution for shared read-only blocks. For all commercial workloads, the top 20% of blocks account for over 90% of requests and the

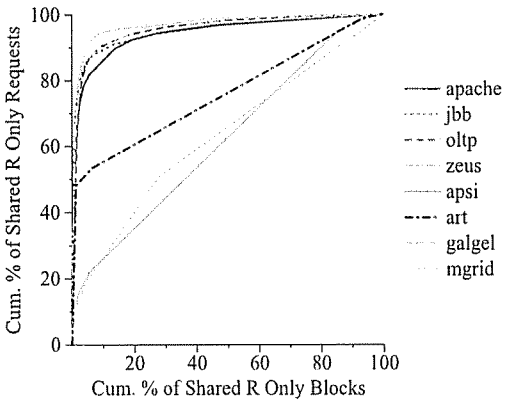

Figure 1. a) Request to Block Distribution: Shared Read-Only Data

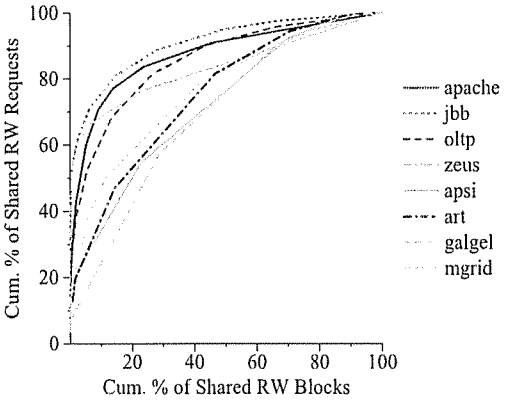

Figure 1. b) Request to Block Distribution: Shared Read-Write Data

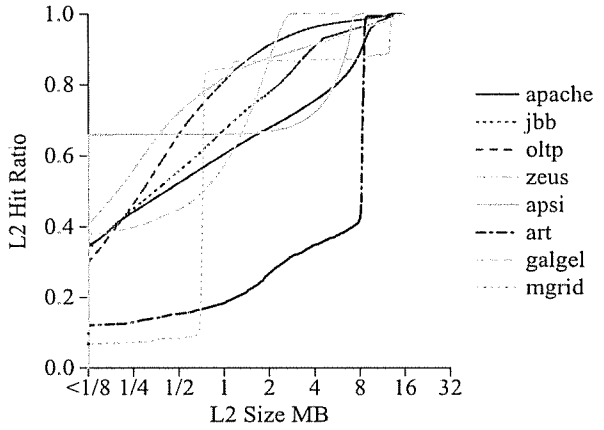

Figure 2. L2 Cache Hit Ratios

top 3% of blocks account for over 70% of requests. Conversely, Figure 1b illustrates that shared read-write blocks have much less locality: the top 20% of blocks only account for 75% or less of requests.

For this reason, we focus on replicating shared read-only blocks in this paper. Further observation (not shown) shows that the top 3% of shared read-only blocks only consume 100-300 KB. Thus, replicating these blocks would have relatively little impact on the total cache capacity.

2.2 Impact of Replication

While replicating blocks can reduce L2 hit latency, it also decreases the effective L2 cache size. If replicas displace too much of a workload's working set, performance may degrade significantly. Figure 2 illustrates this risk by plotting the hit ratios for fully-associative caches up to 32 MB.

$$\text{L2 Hit Ratio} = \frac{\text{Hits within cache size } \alpha}{\text{Hits within a 32 MB L2 cache}}$$

While the L2 Hit Ratio doesn't show the exact change in hits for a practical set-associative L2 cache, it demonstrates the sensitivity that many workloads have to small changes in cache size.

For example, Mgrid and Art have critical working set sizes of 3/4 MB and 8 MB, respectively. Decreasing the available cache capacity below those thresholds has a dramatic negative impact on performance. All of the scientific workloads exhibit clearly identifiable working set boundaries, while the commercial workloads

Table 2. Memory System Configuration Parameters

split L1 I & D caches	32 KB each, 2-way, 3 cycles	
aggregate L2 cache sizes	16 MB 16-way pseudoLRU [24]	4 MB 16-way pseudoLRU [24]
L1/L2 cache block size	64 Bytes	
memory bank + communication latency	200 + 60 cycles	800 + 60 cycles
memory bandwidth	56 GB/s	
memory size	4 GB of DRAM	

have less pronounced transitions. Ideally, a replication policy for private CMP caches would balance the latency benefits against the capacity and miss rate costs.

3 CMP Parameters and Designs

This section specifies the system parameters and CMP designs evaluated in the paper. First, Section 3.1 details the system parameters assumed in the evaluation. Next Section 3.2 describes the three evaluated CMP organizations: a shared L2 cache, a conventional private L2 cache, and a private L2 cache that restricts duplication between caches. Then Section 4 will describe how ASR adapts to workload behavior and selectively permits replication within the restrictive duplication infrastructure.

3.1 System Parameters

All evaluated designs target an eight-processor CMP, assuming the 45 nm technology generation projected for the year 2010 [10]. Table 2 specifies the memory system configuration including the two parameters that are varied in the evaluation: L2 cache size and memory bank latency. The evaluation employs an in-order processor model and each CMP design assumes approximately 310 mm^2 of available die area [10]. We estimate eight processors would occupy 210 mm^2 [21] and eight 2 MB of L2 cache banks with a 20-cycle access latency [1] would occupy 70 mm^2 [10]. The on-chip interconnection network and other miscellaneous structures occupy the remaining area.

All CMP designs in this paper implement the same CMP-Token cache coherence protocol [20]. The token protocol provides a simple and flexible coherence substrate that allows a direct comparison of different replication policies. All designs use writeback, write-allocate caches and implement sequential memory consistency. The intra-chip protocol allows for migratory sharing between L1 caches. The L2 cache is "mostly" inclusive with the L1 caches and maintains up-to-date L1 sharer knowledge. The L2 cache is not strictly inclusive because an L2 block replacement will not invalidate L1 sharers. This optimization saves bandwidth and simplifies the L2 cache controller. All evaluated designs also incorporate a strided prefetcher between the L1 and L2 caches, as well between the L2 caches and memory. The prefetcher is based on the IBM Power 4 [30], except it issues prefetches for both load and stores.

3.2 CMP Designs

CMP-Shared. As illustrated in Figure 3a, the CMP-Shared design assumes a Non-Uniform Cache Architecture (NUCA) [16]. The eight processor cores are arranged with four on an edge, with the chip's center

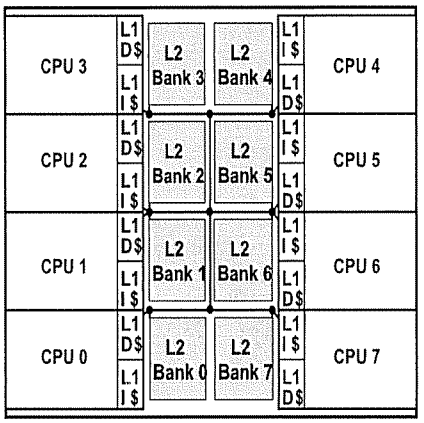

Figure 3. a) Layout of CMP-Shared

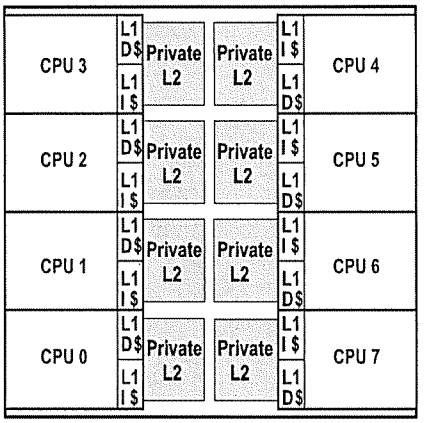

Figure 4. b) Layout of CMP-Private

occupied by eight L2 cache banks. Each processor's L1 I and L1 D caches interface with the packet-switched grid interconnection network. CMP-Shared statically maps the addresses across all eight 2MB cache banks, thus forming a shared 16MB L2 cache with non-uniform access latency. On an L1 cache miss, a processor sends the request across the on-chip network to the appropriate L2 bank. The shared L2 cache offers the best capacity because replication only occurs in L1 caches. However, the shared L2 cache does not exploit the distance locality between a processor and its closest bank.

CMP-Private. In contrast to the CMP-Shared design, the CMP-Private design (Figure 4b) allocates each 2MB cache bank private to a processor. Similar to the Itanium 2 microprocessor [23], the closely integrated private L2 caches allow each processor to avoid the shared on-chip network and directly query the L2 cache tags in parallel with an L1 cache access. L1 misses and replacements are always directed to the local private L2 cache and other processors cannot allocate data into a remote L2 bank. Thus, CMP-Private inherently migrates single requestor data to the requesting processor [4, 13], but their unrestricted replication of shared data can increase off-chip misses and coherence invalidations.

CMP-PrivateCS. To save valuable on-chip storage capacity by preventing unwanted replication between private L2 caches, CMP-Private Conserve Storage (CMP-PrivateCS) implements a ring writeback mechanism [26]. Naively, a private CMP cache design allows replication when separate L1 caches writeback to different local L2 banks. In contrast, CMP-PrivateCS prevents unwanted replication by merging L1 cache writebacks with existing remote L2 copies. Specifically, L1 writeback messages are passed clockwise between private L2 caches to search for an already allocated version of its tag or empty L2 block. The result is ring writebacks allow CMP-PrivateCS to conserve capacity like the CMP-Shared design.

By preventing undesired replications, the CMP-PrivateCS design serves as the basis to evaluate different replication mechanisms. By default, CMP-PrivateCS allows shared read-only data to be replicated, but prevents replicating shared read-write data. CMP-PrivateCS can also be extended with the more selective replication policies. For instance, the initial Victim Replication policy [33] was for an on-chip directory protocol that disallowed replications when the local L2 cache set was filled with home blocks with remote sharers.

We combined Victim Replication with CMP-PrivateCS's token broadcast protocol [20] by disallowing replications when the local cache set was filled with owner blocks with identified on-chip sharers. Another previously proposed replication policy, CMP-NuRapid [6], was to only allocate the local L2 tag after the first request and then locally allocate the actual L2 data block upon a second request. We incorporated CMP-NuRapid's replication policy to CMP-PrivateCS by storing a per processor request count for each L2 block and locally allocating the L2 block as soon as the processor's request count is two. While both previous policies restrict replication using fixed, static criteria, the next section describes our Adaptive Selective Replication mechanism that dynamically adjusts the replication policy to match the current workload behavior and system configuration.

4 Adaptive Selective Replication: ASR

This section describes how Adaptive Selective Replication (ASR) adapts the CMP-PrivateCS cache hierarchy to best meet workload demands and system constraints. The section begins with Section 4.1 describing how replication effects the performance of the memory system. Section 4.2 explains how ASR dynamically analyzes the benefit and cost of replication and adjusts the L2 cache capacity devoted to replica blocks, based on the analysis. Section 4.3 details the ASR implementation.

4.1 Replication and Memory System Performance

Replication will improve memory system performance when the benefit of increasing local L2 cache hits outweighs the cost of increasing off-chip misses. The combination of local L2 hits, remote L2 hits, and off-chip misses determine the average latency of L1 cache misses. For an in-order blocking processor model, the following equation describes the average cycles for L1 cache misses normalized by instructions executed:

$$\text{Memory cycles / Instr.} = \frac{(P_{localL2} \times L_{localL2}) + (P_{remoteL2} \times L_{remoteL2}) + (P_{miss} \times L_{miss})}{Instructions}$$

P_x is the probability of a memory request being satisfied by the entity x, where x is a local L2 cache, the remote L2 caches, or main memory and L_x equals the latency of each entity. Therefore, the combination of the first two localL2 and remoteL2 term represents the memory cycles spent on L2 cache hits and the third miss term depicts the memory cycles spent on L2 cache misses. By replicating more remote blocks locally, L1 misses are more likely to hit in the local L2 cache, thus the $P_{localL2}$ term increases and the $P_{remoteL2}$ term decreases. Because the latency of a local L2 cache hit is tens of cycles faster than a remote L2 cache hit, the net effect of increasing replication will be a reduction in total memory cycles spent on L2 cache hits. However, more replication devotes more capacity to replica blocks, thus fewer unique blocks exist on-chip, increasing the probability of L2 cache misses, P_{miss}. If the probability of a miss increases significantly due to replication, the miss term will dominate, as the latency of memory is hundreds of cycles greater than the L2 hit latencies. Therefore, balancing these three terms is necessary to improve memory system performance.

The optimal degree of replication often lies between allowing all replications and no replications. Figure 4 graphically depicts the general tradeoff of adjusting replication within a CMP cache. The left-hand plot of Figure 4, the L2 hit cycles-per-instruction curve, summaries the trend that increasing the L2 capacity

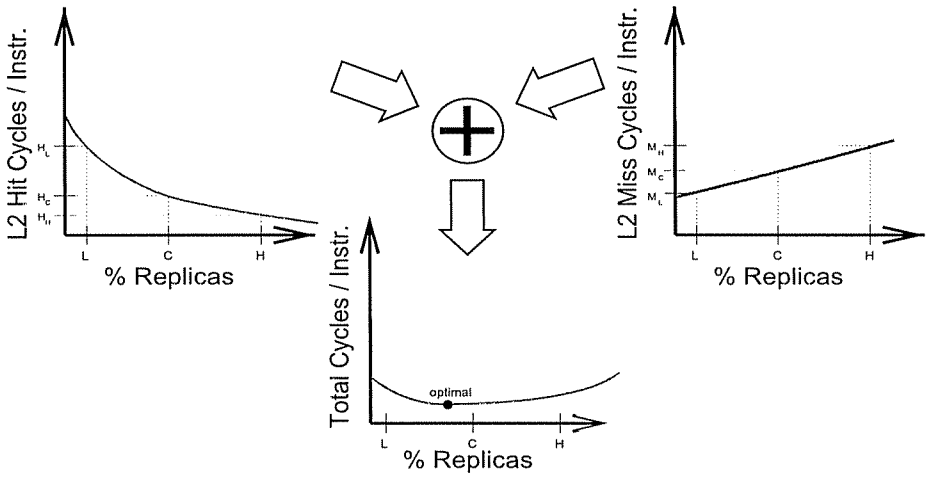

Figure 4. Replication Effecting Memory System Performance

devoted to replication reduces the total memory cycles for L2 cache hits. Due the strong locality of shared read-only requests, a small percentage of L2 replication capacity can significantly reduce L2 hit cycles by allowing many previous remote L2 hits to become local L2 hits. In contrast, a large percentage L2 replication capacity more gradually reduces L2 hit cycles because fewer unique blocks on-chip lead to fewer total L2 hits. The right-hand plot of Figure 4, the L2 miss cycles-per-instruction curve, illustrates that increasing the L2 capacity consumed by replication increases the total memory cycles for off-chip misses. In essence, the L2 miss cycles plot is a segment of the previously presented L2 hit ratio curve, Figure 2, near the current effective L2 cache capacity. Thus the L2 miss cycles plot indicates the value of the unique blocks that have been recently evicted or soon will be evicted by replication. The combination of these two curves, the bottom plot of Figure 4, charts the normalized memory cycle curve and graphically illustrates replication's effect on the memory cycles-per-instruction equation. Due to the fact that the L2 hit cycles curve decreases with increasing replication capacity, and the L2 miss cycles curve increases with increasing replicating capacity, the minimum value of the total memory cycle curve often lies between allowing all replications and no replications. However, finding the specific optimal point of replication for a given set of workload and system parameters requires detailed knowledge of replication's effect on memory system performance.

4.2 Balancing Replication via ASR

By dynamically monitoring the benefit and cost of replication, ASR attempts to achieve the optimal level of replication within the CMP cache hierarchy. Analyzing the entire memory cycles-per-instruction curve to precisely determine the optimal amount of replication for a set of workload and system parameters is prohibitively expensive, especially in hardware. Instead, ASR breaks down the normalized memory cycle curve into a series of distinct points. Then ASR simplifies the analysis to a *local* decision of whether the amount of replication should be increased, decreased, or remain the same. Figure 4 illustrates the case where the current replication level, labeled C, results in H_C hit cycles-per-instruction and M_C miss cycles-per-instruction. ASR considers three alternatives: increasing replication to the next higher level, labeled H, decreasing repli-

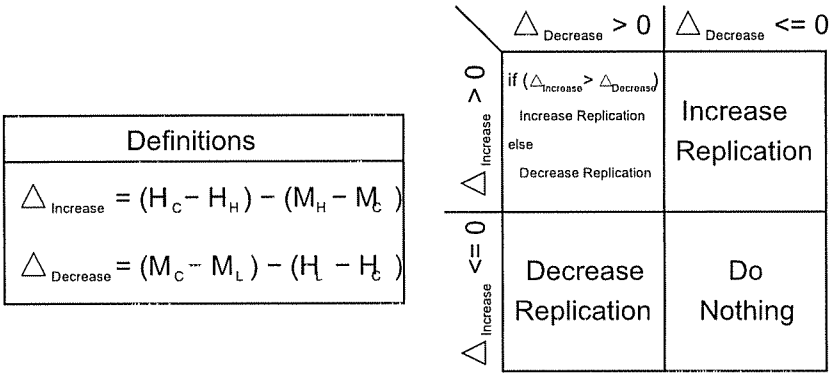

Figure 5. ASR Decision Table for Adjusting Replication

cation to the next lower level, labeled L, or leaving the replication level unchanged. To make this decision, ASR not only needs H_C and M_C, but also four additional hit and miss cycles-per-instruction values: H_H and M_H for the next higher level and H_L and M_L for the next lower level.

To simplify the collection process, ASR estimates only the four differences between the hit and miss cycles-per-instruction: 1. the decrease of L2 hit cycles caused by increasing replication, $(H_C - H_H)$; 2. the increase of L2 hit cycles caused by decreasing replication, $(H_L - H_C)$; and 3. the increase of L2 miss cycles caused by increasing replication, $(M_H - M_C)$; and 4. the decrease of L2 miss cycles caused by decreasing replication, $(M_C - M_L)$.

By comparing the difference between $(H_C - H_H)$ and $(M_H - M_C)$ and the difference between $(H_L - H_C)$ and $(M_C - M_L)$, ASR will increase, decrease, or maintain constant the amount of replication. Figure 5 presents the decision table ASR uses to adjust the amount of replication. When both $\Delta_{Increase}$ and $\Delta_{Decrease}$ agree the benefits for increasing or decreasing replication exceed the costs, the amount of replication is adjusted accordingly. Similarly, when both $\Delta_{Increase}$ and $\Delta_{Decrease}$ indicate the costs of adjusting the amount of replication outweigh the benefits, ASR maintains the current level of replication. However, if $\Delta_{Increase}$ and $\Delta_{Decrease}$ disagree in the direction replication should be changed, ASR chooses the direction with the greater estimated benefit.

4.3 Implementing ASR

This section describes the details of Adaptive Selective Replication. The section begins with the storage required to identify shared read-only blocks, then continues with the description of ASR's process for replicating frequently requested blocks. Finally, the section ends with how ASR determines the replication's effect on memory system performance.

4.3.1 Identifying Shared Read-Only Blocks

To identify which cache blocks are shared and read-only, ASR uses a per-block dirty bit in combination with a per-block shared bit. The dirty bit is already used by current systems to identify which blocks are modified with respect to memory, and must be written back on replacement. The L1 and L2 cache tags also include a shared bit that is set when receiving a coherence request from a different processor than the current sharer.

Table 3. ASR Replication Levels

Replication Level	0	1	2	3	4	5	6	7	8	9
Probability of replication	0	1/1024	1/256	1/64	1/16	1/8	1/4	1/2	3/4	1

Similar to the dirty bit, once the shared bit set, it is not reset until the block is replaced to memory. When neither the dirty bit nor shared bit is set, the block is considered shared read-only.

4.3.2 Replicating Frequently Requested Blocks

To simplify the replication process, ASR breaks down the amount of replication into distinct steps. ASR has ten replication levels (Table 3) that permit various levels of replication within the local L2 cache. Each replication level has a unique probability that a shared read-only block will be replicated, with the lower replication levels permitting very few replications. In effect, the probabilistic policy biases replications to the most frequently requested L2 blocks because each request increases the probability the block is replicated upon its subsequent L1 cache eviction. Once replicated locally, there may be many more future requests for the block that will encounter the fast local L2 hit latency rather than the slow remote L2 hit latency. Therefore, through selective probabilistic replication, ASR exploits the request locality of shared read-only data to provide most of replication's latency benefit, while avoiding the cost of replicating every shared read-only block.

ASR further exploits the locality of shared read-only requests by biasing replications to the frequently requested blocks that lie on the top of L2 cache sets' pseudo LRU stacks [12]. ASR includes two extra bits in the L1 cache tags to store whether the block's L2 LRU position was in one of four ranges of LRU stack depth: '0', '1-7', '8-14', '15'. Blocks received from a L1 cache or memory are set to range '0'. Upon L1 cache eviction, the replication decisions of range '0' blocks are based solely on the probability of replication, but replication decisions for the other three ranges, '1-7', '8-14', '15', are further filtered by the ratios of 0.75, 0.50, and 0.25 respectively.

4.3.3 Measuring Replication Benefit and Cost

Determining the change in the replication benefit and cost requires identifying those replications that would or would not have occurred with the next higher and lower replication levels. As previously stated, ASR uses distinct replication levels to discretely adjust the probability that a block is replicated locally. Specifically, ASR utilizes a linear feedback shift register as a pseudo-random number generator [11]. ASR uses the random numbers, along with the current replication level's probability, to determine whether a replication will occur. By modulating the random numbers by decreasing powers of 2, the replications occurring at replication level n, also occur at the next higher level n+1, plus some additional replications for the next higher level. The subsumption of lower level replications allows ASR to distinguish the distinct groups of cache blocks that will or will not be replicated for the next higher and lower levels respectively.

The remainder of this section describes ASR's four separate mechanisms for estimating the costs and benefits of increasing and decreasing replication, as well as, what triggers a replication analysis.

The Cost of Decreasing Replication (H_L - H_C). To estimate the cost of decreasing replication, ASR marks the blocks that are replicated with the current replication level, but not with the next lower level. Spe-

cifically, an extra *current replication* bit marks these blocks in the local L2 cache tags. For local L2 hits that find the current replication bit set, ASR increments the (H_L - H_C) counter by latency benefit of a local L2 hit versus a remote L2 hit (30 cycles). When the replication level is increased, the current replication bits are cleared because they no longer correspond to the blocks of the new lower replication level group.

The Benefit of Increasing Replication (H_C - H_H). To determine the benefit of increasing replication, ASR tracks the blocks that are not replicated at the current replication level, but would have been with the next higher level. Specifically, ASR identifies these blocks using separate Next Level Hit Buffers (NLHBs) for each L2 cache. The NLHB stores the 8-bit partial tags [15] of blocks that would have been replicated with the next higher replication level. To record most blocks that would be replicated with the next higher level, NLHB is sized to a 16 K entry, 16-way set associative buffer. When a request hits in a remote L2 cache, the NLHB is checked to determine if the request could have been a local hit if replication was increased. If so, ASR increments the (H_C - H_H) counter by the 30-cycle local L2 hit latency benefit.

The Cost of Increasing Replication (M_H - M_C). ASR approximates the increase in miss cycles associated with increasing the replication level by estimating the utilization of the L2 cache blocks soon to be evicted. While the NLHB must store 16 K entries of partial tags to determine the benefit of the next higher replication level, ASR only needs to monitor the utilization of the last 1 K of least recently used L2 blocks to determine the cost of the next higher replication level. Due to the low locality of these infrequently used L2 cache blocks, there is virtually no additional benefit for increasing the monitor size past 1 K. Because precisely determining the recently used cache blocks is prohibitively expensive in hardware, ASR uses way and set counters [28] to estimate which blocks are the least recently used. Specifically, ASR breaks the L2 sets into 256 separate groups using the high order L2 cache index bits and relies on a 255-bit pseudo LRU binary tree [24] to estimate the LRU position of the requested set group versus the other set groups. If an on-chip request hits a L2 block that is not identified as a current replica, and the combination of the L2 block's set group and way LRU position lies within the last 1 K of L2 blocks, the (M_H - M_C) counter is incremented by the off-chip memory latency.

The Benefit of Decreasing Replication (M_C - M_L). To predict the benefit of decreasing replication, ASR uses Victim Tag Buffers (VTBs) to track which L2 misses could have been avoided by reducing the replication level. Each separate ASR uses the 1K entry 16-way set associative buffer to store the most recently evicted block's 16-bit partial tags from their associated L2 cache. The VTB only stores blocks that were evicted due to the current replication level, but would not have been with the next lower level. When an L2 eviction occurs that is identified to be caused by the current replication level, the evicted block is allocated into the VTB. All other L2 evictions are only stored in the VTB if they replace an existing valid entry. Subsequent off-chip misses that hit in the VTB, increment the (M_C - M_L) counter by the off-chip miss latency. When the replication level is decreased, the victim buffer is cleared because the blocks currently in the VTB don't estimate the off-chip misses caused by the new lower replication level. Overall, by utilizing

Table 4. Rules for Triggering Replication Evaluation

When one of the two values > 1 K entry monitor size
1. **Number of local L2 replications** that would not have happened **with the next *lower* replication level**
2. **Number of remote L2 replacements** that would have been replicated **with the next *higher* replication level**

Table 5. Adaptive Selective Replication Storage Overhead

Overhead	Bits	K Entries	Total KBytes
per L1 block	3	8	3
per L2 block	2	256	64
next-level hit buffer	8	128	128
victim tag buffer	16	8	16
		Total KBytes	211
		% of Total On-chip Cache Capacity	1.2%

these four counters, ASR can determine whether to increase or decrease replication. However, constantly evaluating the counters is inefficient. A key aspect of the ASR is deciding when to evaluate the counters.

Triggering a Cost-Benefit Analysis. ASR triggers an analysis of the four benefit and cost counters after observing enough events to ensure a fair comparison. Table 4 shows the two rules ASR uses for triggering an analysis of recent replication opportunities. By counting the local L2 replications and remote L2 replacements, the time interval between replication analyses is not fixed, but rather depends on the frequency of replication opportunities observed by ASR. Upon triggering a replication evaluation, ASR performs the comparison described in Section 4.2 to determine if and how the replication level should be changed. Four consecutive evaluation results in the same direction causes an actual change the replication level. The four-evaluation-result hysteresis ensures ASR has seen a definite change in workload behavior before changing the replication level, thus preventing unnecessary replication level adjustments. After each ASR evaluation, all four counters are cleared.

Hardware Summary. Overall, the storage space required by ASR is approximately 1.2% of the total on-chip cache capacity. Table 5 breaks down the storage requirement of the ASR's main components. Because the ASR only needs to estimate the impact of replication and has no effect on correctness, the ASR uses partial tags in the NLHBs, and VTBs to further reduce the state requirements. ASR's replication logic lies on the non-latency critical L1 replacement decision, and is a simple probabilistic choice. While ASR costs bits, it doesn't cost bandwidth to pass messages between processors to coordinate replication level changes. All information required by each ASR instantiation is stored locally within each CMP node.

5 Methodology

We evaluate all configurations using full-system simulation based on Virtutech Simics and the Wisconsin GEMS toolset [32]. GEMS extends Simics to provide detailed timing of a CMP memory hierarchy. The intra-chip and inter-chip networks are modeled in detail, including all messages required to implement the

Table 6. Evaluation Methodology

Bench	Fast Forward	Warm-up	Executed
Commercial Workloads (unit = transactions)			
apache	2000000	2000	1000
jbb	200000	15000	10000
oltp	100000	300	200
zeus	2000000	2000	2000
Scientific Workloads (unit = billion instructions)			
apsi	89	4.6	loop completion
art	121	3.2	loop completion
galgel	235	3.4	loop completion
mgrid	33	3.0	loop completion

coherence protocol. Virtual cut-through routing is used with two message buffering at all switches except the buffers between the on-chip and off-chip networks are extended to 20 entries to decouple the on-chip network from off-chip queueing delay.

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 SPECOMP benchmarks [3]: Apsi, Art, Galgel and Mgrid. 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 6. 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. All simulations contain small pseudo-random perturbations in the memory latency to account for the non-determinism that exists in multi-threaded workloads. The error bars shown in Section 6.3 indicate the results 95% confidence interval.

6 Evaluation

6.1 Replication Capacity and Memory Cycles

The optimal point of replication shifts depending on workload behavior and system constraints. Figure 6 displays the L2 hit cycles-per-instruction, L2 miss cycles-per-instruction, and the Total cycles-per-instruction curves for three sets of system configurations. Each point on the curve corresponds to a static ASR replication level. The first row of graphs, Figure 6 a-c, presents the cycles-per-instruction curves for a CMP with a large, 16 MB aggregate L2 cache capacity, referred to as the *16 MB 200 lat.* configuration. In this case, replication exploits the large cache size and effectively reduces the L2 hit cycles-per-instruction by 0.4 to 0.6 for Apache, Oltp, Zeus, and Art. In contrast, increasing replication capacity only increase Apache's and Art's L2 miss cycles-per-instruction by greater than 0.2. Therefore, the resulting Total cycles-per-instruction plot, Figure 6c, reveals that the optimal point of replication for six of the eight workloads is near maximum

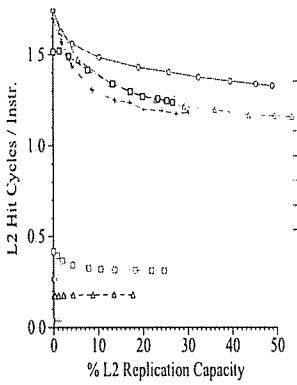

Figure 6. a) L2 Hit Cycles / Instr. (16 MB 200 lat.)

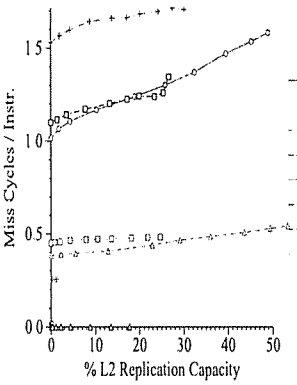

Figure 6. b) L2 Miss Cycles / Instr. (16 MB 200 lat.)

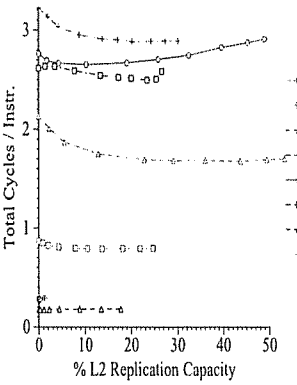

Figure 6. c) Total Cycles / Instr. (16 MB 200 lat.)

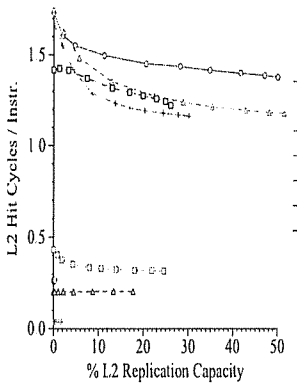

Figure 6. d) L2 Hit Cycles / Instr. (16 MB 800 lat.)

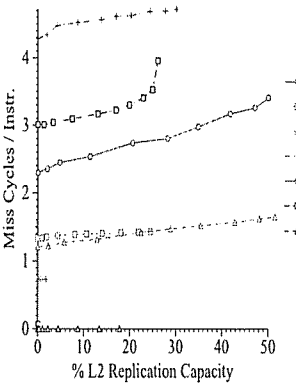

Figure 6. e) L2 Miss Cycles / Instr. (16 MB 800 lat.)

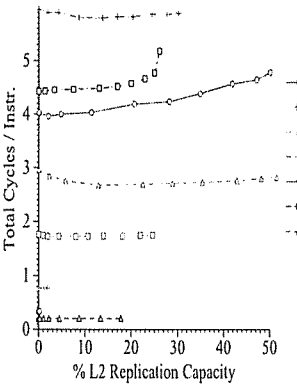

Figure 6. f) Total Cycles / Instr. (16 MB 800 lat.)

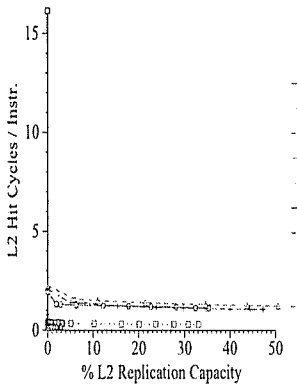

Figure 6. g)L2 Hit Cycles / Instr. (4 MB 800 lat.)

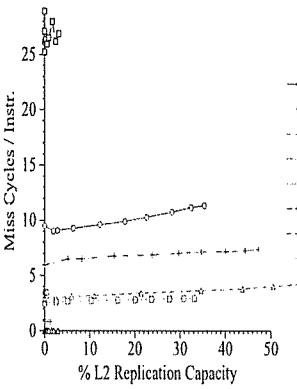

Figure 6. h) L2 Miss Cycles / Instr. (4 MB 800 lat.)

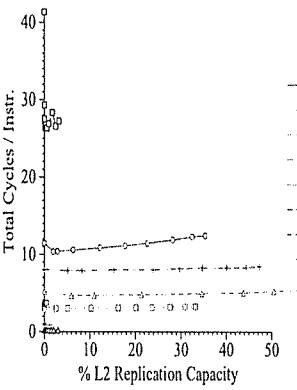

Figure 6. i) Total Cycles / Instr. (4 MB 800 lat.)

replication. While the remaining two workloads, Apache and Art, prefer a replication capacity considerably less than the maximum.

With slower memory latencies, the second row of memory curves Figure 6 d-f show that the optimal level of replication shifts towards less replication. We evaluated a CMP with an 800-cycle memory latency (i.e. the *16 MB 800 lat.* configuration) to study how the longer memory latencies effect the optimal level of replication. Due to the same aggregate L2 cache size, the L2 hit cycle curves of Figure 6d, maintain the same basic

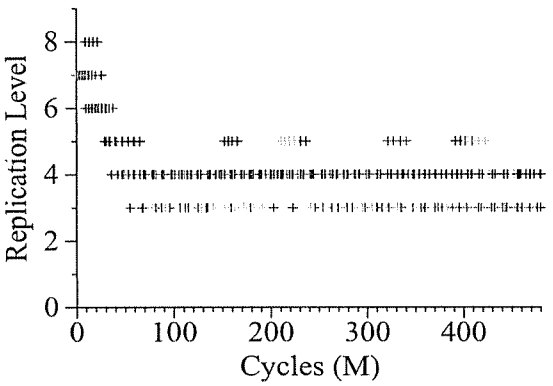

Figure 7. a) Apache Replication Change (16 MB 200 lat.)

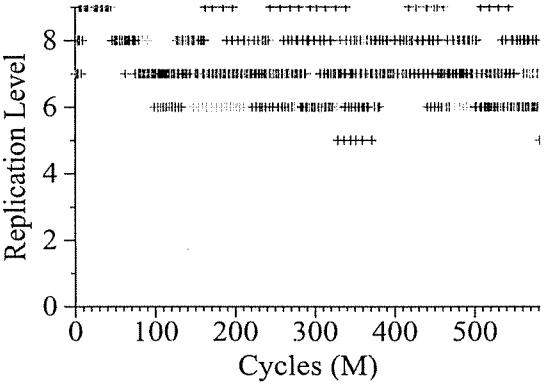

Figure 7. b) Oltp Replication Change (16 MB 200 lat.)

shape as those of Figure 6a. However, the slower memory latency causes the L2 miss cycle curves in Figure 6e to substantially increase with respect to those in Figure 6b. The result is the miss cycle curves have a greater impact on the total cycle curves. For instance the optimal point of replication capacity for Oltp shifted from near 40%, for the previous 200 memory cycle configuration, to close to 20% for the 800 memory cycle configuration.

Figure 6 g-i display how a smaller aggregate L2 cache size further shifts the optimal level of replication towards less replication. We studied the 4 MB aggregate L2 cache size configuration, referred to as *4 MB 800 lat.*, to account for the scaled down working sets of the evaluated workloads [2]. The smaller L2 capacity cause the L2 hit cycles-per-instruction to decrease and the miss cycles-per-instruction to increase for all workloads. Art exhibits the most significant change because its 8 MB working set (Figure 2) no longer fits in the on-chip cache. The resulting total cycles-per-instruction graph shows that replica blocks can consume up to 53% of all L2 capacity in the four commercial workloads, but that the optimal replication level for all workloads is near 0%.

6.2 Adapting to Workload Behavior

By dynamically monitoring the changes in L2 hit cycles and L2 miss cycles, ASR matches the level of replication within each local L2 cache to the behavior of each individual processor. Figure 7 illustrates ASR's dynamic adjustment of each private L2 cache's replication level over the runtime of the workload. Both plots of Figure 7 use the 16 MB 200 lat. configuration and all ASR replication levels are initialized to level 7. Each point on the plots indicates when an ASR triggered an evaluation of its counters (x-axis) and the current replication level (y-axis).

For the workload Apache, ASR reduces the replication level to achieve the lower replication capacity preferred by the workload. Figure 7a demonstrates that each ASR replication level drops within the first 40 million cycles to the replication levels 3-5 and stays in this range for the remainder of the execution. The result is the average dynamic L2 capacity consumed by replicas in Apache is 15%.

For the workload Oltp, which has an optimal point of replication capacity between 25-50%, ASR adjusts replication to between the levels 5 and 9. Figure 7b illustrates initially each processor's ASR mechanism

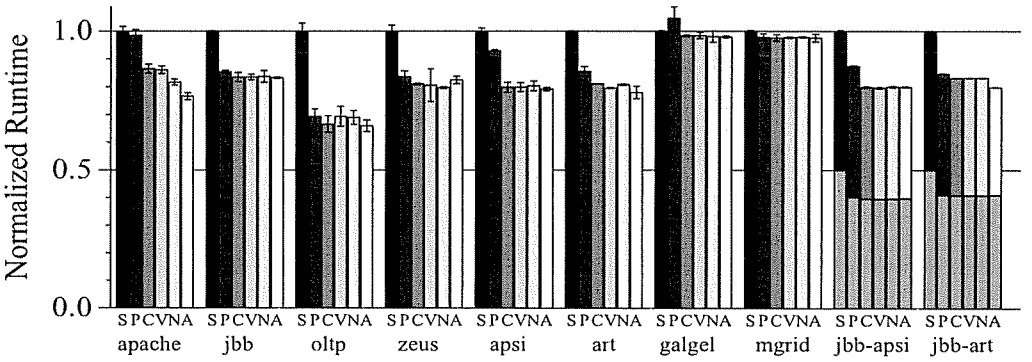

Figure 8. Normalized Runtimes (16 MB 200 lat.) S: CMP-Shared, P: CMP-Private, C: CMP-PrivateCS, V: Victim Replication, N: CMP-NuRapid, A: ASR

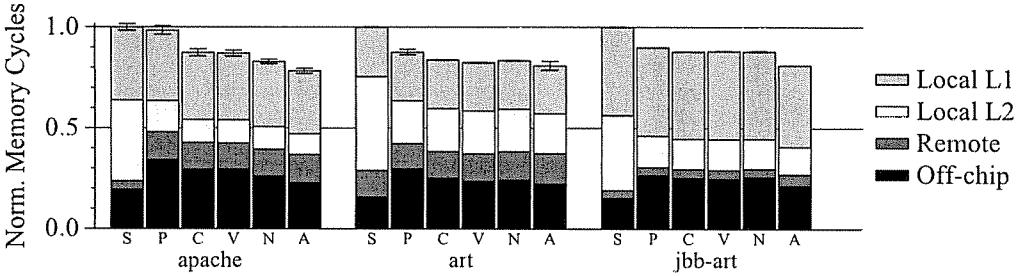

Figure 9. Memory System Cycles (16 MB 200 lat.) S: CMP-Shared, P: CMP-Private, C: CMP-PrivateCS, V: Victim Replication, N: CMP-NuRapid, A: ASR

quickly detects a benefit for replicating Oltp's large instruction footprint and moves all eight L2 caches to level 9, or 100% probability of replication within the first 20 million cycles. Then each ASR compensates for the overly aggressive replication level, and moves all 8 caches back to level 7 by the 80 million cycle mark. Finally, ASR falls into a steady state where each cache's replication level varies between levels 5 and 9 for the remainder of the run, resulting in a 41% average L2 capacity consumed by replicas.

In order to remove cold-start effects, Section 6.3 evaluates ASR by initializing the ASR replication levels to their steady-state values.

6.3 Comparison of Replication Schemes

By matching the level of replication to the workload behavior and system parameters, ASR outperforms the baselines CMP-Shared, CMP-Private, and CMP-PrivateCS, as well as the previously proposed replication policies of Victim Replication [33] and CMP-NuRapid [6]. Figure 8 shows the normalized runtime of each CMP design executing the 8 homogeneous workloads, plus 2 heterogeneous mixtures Jbb-Apsi and Jbb-Art. Heterogeneous bars are split with the top sections indicating the normalized cycles per transaction for Apsi or Art and the bottom indicating the normalized cycles per transaction for Jbb. The private cache designs exploit the high capacity and relative fast memory latency of the 16 MB 200 lat. configuration and achieve a 2% to 31% reduction in runtimes versus the shared cache design. As forecasted by the total cycles-per-instruction curve, Figure 6c, the three private cache designs that restrict shared read-only replication (Victim Replication, CMP-NuRapid, and ASR) attain better performance for workloads Apache and Art than the two

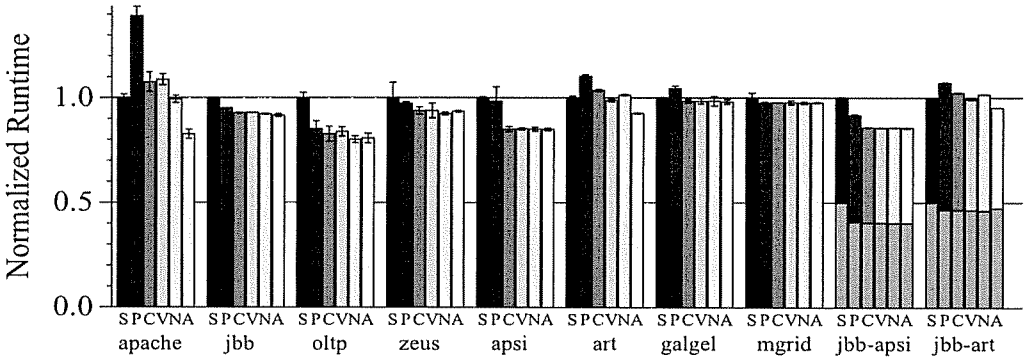

Figure 10. Normalized Runtimes (16 MB 800 lat.) S: CMP-Shared, P: CMP-Private, C: CMP-PrivateCS, V: Victim Replication, N: CMP-NuRapid, A: ASR

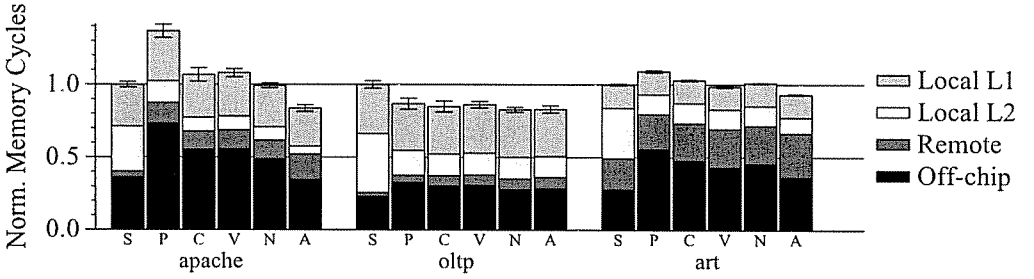

Figure 11. Memory System Cycles (16 MB 800 lat.) S: CMP-Shared, P: CMP-Private, C: CMP-PrivateCS, V: Victim Replication, N: CMP-NuRapid, A: ASR

private cache designs that allow all shared read-only replication (CMP-Private and CMP-PrivateCS). ASR attains the best performance and reduces runtimes versus the second best replication policy, CMP-NuRapid, by 11%, 5%, and 8% for Apache, Art, and Jbb-Art respectively.

To provider further insight, Figure 9 shows the memory system cycle breakdown for the Apache and Art workloads to indicate where the time is spent in the memory system. The 'Local L1' and 'Local L2' segments display the fraction of the average memory access time contributed by local L1 and L2 hits respectively (for CMP-Shared 'Local L2' includes all L2 hits). The 'Remote' bar segment represents the cycles spent on requests satisfied by remote L1 or L2 caches. Finally, the 'Off-chip' bar segment exposes the cycles spent on off-chip misses.

By devoting 61% less L2 capacity to replications than CMP-NuRapid (the next most restrictive replication proposal), ASR achieves a similar contribution for off-chip miss cycles as CMP-Shared when running Apache. Meanwhile, the data ASR does replicate in Apache, enables it to achieve a similar sum for the 'Local L2' and 'Remote' cycles as the CMP-PrivateCS design. For the heterogeneous mixture of Jbb and Art, ASR exhibits the same performance for Jbb as the other private cache designs, but reduces the cycles to reach the end of the main loop in Art by 8% versus CMP-NuRapid.

ASR's flexibility allows it to adapt when the memory latency is increased. In Figure 10, the memory latency is increased to 800 cycles, thus increasing the penalty for off-chip misses. The higher memory latency diminishes the advantage the private cache designs have over CMP-Shared. While ASR maintains it perfor-

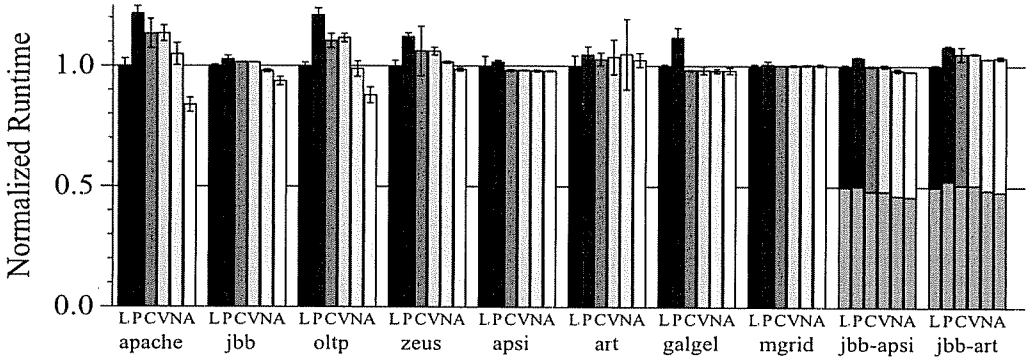

Figure 12. Normalized Runtimes (4 MB 800 lat.) L: CMP-Shared, P: CMP-Private, C: CMP-PrivateCS, V: Victim Replication, N: CMP-NuRapid, A: ASR

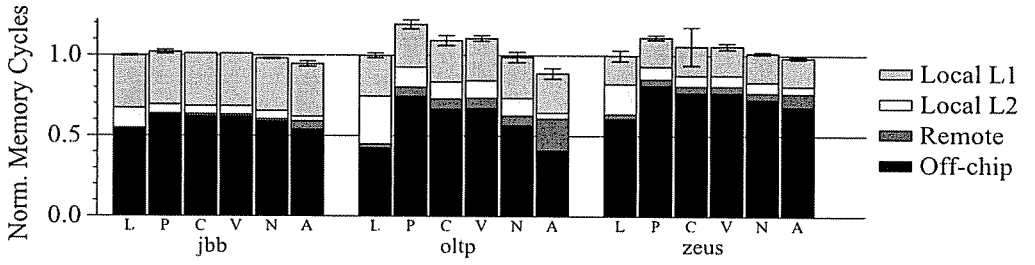

Figure 13. Memory System Cycles (4 MB 800 lat.) L: CMP-Shared, P: CMP-Private, C: CMP-PrivateCS, V: Victim Replication, N: CMP-NuRapid, A: ASR

mance improvement over the CMP-Shared design for the workloads Apache, Art, and Jbb-Art, the other private cache designs actually perform worse than CMP-Shared. Furthermore, by limiting the average L2 replication capacity to 23%, ASR reduces the runtime for Oltp by 5% versus the second best private cache design. Figure 11 breaks down the memory cycles for Oltp, Apache, and Art. By comparing the Oltp memory cycles in ASR to that of CMP-Shared, it is evident that ASR does allow enough replications such that off-chip cycles increase by 16% versus CMP-Shared. However the tradeoff in capacity allows ASR to reduce the 'Local L2' and 'Remote' cycles by 49% versus CMP-Shared.

For most workloads, the advantage of ASR's adaptability is fully exemplified when the cache capacity is constrained. To account for scaled down commercial workloads [2], the total L2 cache is reduced to 4 MB in Figure 12. Unlike the 16MB configuration, most 4 MB private cache designs perform worse than the CMP-Shared design when running commercial workloads. However, ASR matches or exceeds CMP-Shared's performance for all workloads except Jbb-Art, where it encounters a 1% increase in cycles per transaction. In this heterogeneous workload, a global L2 cache allocation policy is preferred because it allows Art's larger working set to consume more cache capacity. The ASR's associated with processors running Art train to replication level 0, but ASR still encounters 44% more misses than CMP-Shared because Art's large data set cannot be allocated into the other private L2 caches running Jbb.

For the commercial workloads Apache, Oltp, and Zeus the restrictive replication policies of Victim Replication and CMP-NuRapid increase the runtimes by 5% to 13% versus CMP-Shared. In contrast ASR adapts

the Private CMP cache hierarchy to severely restrict replication, thus for these same three commercial workloads ASR achieves a 2% to 17% reduction in runtime versus CMP-Shared and a 3% to 21% reduction in runtime versus the second best private CMP design, CMP-NuRapid. Figure 13 breaks down the cycles for Jbb, Oltp, and Zeus to show that the ASR performance improvement is due to achieving similar off-chip cycles as CMP-Shared, while exploiting the local private L2 cache latency to reduce on-chip cycles.

7 Related Work

7.1 Multiprocessor Memories

A large body of previous work exists in studying data replication in the context of flat multiprocessors [7]. Specifically, throughout the previous decade significant work has compared hardware solutions such as CC-NUMA and Flat COMA architectures [27, 34], along with software [5, 31] and hybrid hardware/software combinations [9, 25]. The Flat COMA protocol [25, 27] removed the slow ordered network of hierarchical COMA machines allowing data to migrate and replicate towards the requesting processor on an unordered network, similar to the evaluated private CMP cache designs. Related to ASR's adaptive selective replication mechanism, Verghese et al. [31] proposed an OS mechanism that adapted the number of pages migrated and replicated to a processor's local memory. Zhang et al [34] studied the working sets of various workloads running on a NUMA system, and comparable to our work in Section 2, they characterized data into three classes: replication, migration read-only, and migrating read/write.

7.2 Chip Multiprocessor Caches

There has also been significant recent work in evaluating the benefits and limitations of replication in CMP caches. Huh et al. [13] investigated sharing in a CMP-NUCA cache and concluded allowing some replication between cache banks was advantageous. Liu et al. [19] evaluated the performance of managing the allocation of cache resources on a bus-based CMP and proposed a profile-driven approach to determine which cache banks to share between processors and which to reserve as private. In contrast to these proposals, ASR dynamically analyzes workload behavior and adapts L2 cache replication on a per cache block basis to match the current workload demands.

The most closely related proposals to Adaptive Selective Replication are the previously discussed Victim Replication and CMP-NuRapid proposals by Zhang et al. [33] and Chishti et al. [6] respectively. Both designs restrict shared read-only data replications, but their static mechanisms tend to favor certain workloads and fail to adjust to changes in workload behavior and system constraints.

Finally, similar to ASR Suh et al. [29] used set and way counters to monitor cache block utilization. However, Suh et al. used the monitoring information to dynamically partitioned ways in a set-associative cache among multiple threads, while ASR uses the monitoring information to determine the cost of increasing replication.

8 Conclusions and Future Work

Managing on-chip wire delay, while limiting off-chip misses, is essential in order to improve future CMP performance. A private CMP cache hierarchy offers lower access latency than a shared cache, but uncon-

trolled replication may cause significant performance degradation due to increased off-chip misses. In this paper, we observed for commercial workloads, shared read-only data is frequently requested and exhibits high request locality. Then we propose ASR, which dynamically adapts shared read-only data replication to exploit the latency advantage of a private caches without wasting cache capacity due to excessive replication. By performing an opportunity analysis of replication, ASR adjusts the degree of replication to match the current workload behavior and system configuration, the providing a robust CMP cache hierarchy.

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, University of Texas 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] 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.

[4] B. M. Beckmann and D. A. Wood. Managing Wire Delay in Large Chip-Multiprocessor Caches. In *Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2004.

[5] 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.

[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*, June 2005.

[7] F. Dahlgren and J. Torrellas. Cache-Only Memory Architectures. *IEEE Computer*, 32(6):72–79, June 1999.

[8] K. Diefendorff. Power4 Focuses on Memory Bandwidth. *Microprocessor Report*, 13(13):1–8, Oct. 1999.

[9] 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.

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

[11] S. W. Golumb. *Shift Register Sequences*. Aegean Park Press, revised edition, 1982.

[12] M. D. Hill and A. J. Smith. Evaluating Associativity in CPU Caches. *IEEE Transactions on Computers*, 38(12):1612–1630, 1989.

[13] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. A NUCA Substrate for Flexible CMP Cache Sharing. In *Proceedings of the 19th International Conference on Supercomputing*, June 2005.

[14] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway. The AMD Opteron Processor for Multiprocessor Servers. *IEEE Micro*, 23(2):66–76, March-April 2003.

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

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

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

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

[19] 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.

[20] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. K. Martin, and D. A. Wood. Improving Multiple-CMP Systems Using Token Coherence. In *Proceedings of the Eleventh IEEE Symposium on High-Performance Computer Architecture*, Feb. 2005.

[21] H. McIntyre and et al. A 4-MB On-Chip L2 Cache for a 90-nm 1.6-GHz 64-bit Microprocessor. *IEEE Journal of Solid-State Circuits*, 40(1):52–59, Jan 20055.

[22] C. McNairy and R. Bhatia. Montecito: A Dual-Core Dual-Thread Itanium Processor. *IEEE Micro*, 25(2):10–20, March/April 2005.

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

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

[25] 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.

[26] 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*, June 2005.

[27] P. Stenström, T. Joe, and A. Gupta. Comparative Performance Evaluation of Cache-Coherent NUMA and COMA Architectures. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992.

[28] G. E. Suh, S. Devadas, and L. Rudolph. A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning. In *Proceedings of the Eighth IEEE Symposium on High-Performance Computer Architecture*, Feb. 2002.

[29] G. E. Suh, L. Rudolph, and S. Devadas. Dynamic Cache Partitioning for CMP/SMT Systems. *Journal of Supercomputing*, pages 7–26, 2004.

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

[31] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum. Operating System Support for Improving Data Locality on CC-NUMA Compute Servers. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 1996.
[32] Wisconsin Multifacet GEMS Simulator. http://www.cs.wisc.edu/gems/.
[33] M. Zhang and K. Asanovic. Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, June 2005.
[34] Z. Zhang and J. Torrellas. Reducing Remote Conflict Misses: NUMA with Remote Cache versus COMA. In *Proceedings of the Third IEEE Symposium on High-Performance Computer Architecture*, Feb. 1997.