ASR: Adaptive Selective Replication for CMP Caches

Bradford M. Beckmann[†]
Windows Server Performance
Microsoft Corporation
Bradford.Beckmann@microsoft.com

Michael R. Marty and David A. Wood
Computer Sciences Department
University of Wisconsin—Madison
{mikem, 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 off-chip misses. Others use private L2 caches, replicating data to limit the delay due to global wires and minimize cache access time. Recent hybrid proposals use selective replication to balance latency and capacity, but their static replication rules result in performance degradation for some combinations of workloads and system configurations.

This paper proposes Adaptive Selective Replication (ASR), a mechanism that dynamically monitors workload behavior to control replication. ASR replicates cache blocks only when it estimates the benefit of replication (lower L2 hit latency) exceeds the cost (more L2 misses). Full-system simulations of 8-processor CMPs show that ASR provides robust performance: improving performance by as much as 29% versus shared caches, 19% versus private caches, and 12% versus CMP-NuRapid [9] and Victim Replication [41]. Furthermore, while ASR does not improve the performance of all workloads, it provides performance stability by always performing at least comparably to the best alternative including Cooperative Caching [8].

1. Introduction

As Chip Multiprocessors (CMPs) emerge in mainstream systems, they must provide good performance for a wide variety of workloads. 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 5 [27] and Sun Niagara [18], employ shared L2 caches to maximize the on-chip cache capacity by preventing replication. While shared caches usually minimize off-chip misses, they have higher access latencies since many requests cross global wires to reach distant L2 banks. In contrast, private L2 caches [19, 23] reduce average access latency by replicating data close to the requesting processor, but sacrifice effective capacity and incur more misses.

Recent hybrid cache designs seek to achieve a balance between latency and capacity by selectively replicating cache blocks. Cooperative Caching [8] and CMP-NuRapid [9] have nominally private L2 caches and restricts replication under certain criteria, while Victim Replication [41] has a nominal shared L2 cache and allows replication under other criteria. These schemes perform better than private and shared caches for selected workloads and system configurations. However, CMP-NuRapid and Victim Replication each have static replication policies that cannot dynamically adapt to different workload behavior. Cooperative Caching uses a configurable probability to tradeoff replication with effective cache capacity, but does not propose a method to adjust the probability.

Figure 1 illustrates the need for an adaptive replication policy. For the 16 MB CMP configuration (see Section 5.1), Cooperative Caching improves the performance of Apache by 13% using minimum replication (CC 100%), but degrades the performance of Apsi by 27% at the same level. Furthermore, Section 6.2 shows that for some workloads the optimal replication level changes for different cache configurations. Clearly, some adaptive policy is needed to determine the best replication level for a given combination of workload and cache configuration.

This paper proposes *Adaptive Selective Replication (ASR)*, a hardware mechanism that dynamically estimates the cost (extra misses) and benefit (lower hit latency) of replication and adjusts the replication level to minimize average access time. ASR monitors hits to remote L2 cache banks and (pseudo-)LRU cache blocks, to estimate the benefits and costs, respectively, of additional replication. ASR monitors hits to replica blocks and a novel Victim Tag Buffer to estimate the benefit of reducing replication. ASR maintains per-processor summaries of the costs and benefits, allowing independent localized replication decisions.

This paper makes the following contributions:

- We demonstrate that cache replication policies should focus on shared read-only blocks. For commercial workloads, shared read-only blocks account for 42-71% of L2 requests, but consume—without replication—only 10-21% of the L2 capacity. Replicating relatively few shared read-only blocks significantly reduces L2 access time due to their tremendous locality: the top 3% of shared read-only blocks account for 70% of requests. Conversely, aggressive replication degrades some workloads' performance due to increased off-chip misses.
- We introduce *Selective Probabilistic Replication (SPR)*, a simple replication mechanism that exploits the fact that the most frequently requested L2 blocks are also the most frequently evicted L1 blocks. By using probabilistic filtering, SPR requires significantly less hardware than CMP-NuRapid and Cooperative Caching, and equivalent hardware to Victim Replication.

[†]Work performed while a Ph.D. student at Wisconsin.

This work is supported in part by the National Science Foundation (NSF), with grants CNS-0205286 and CCR-0324878, as well as donations from Intel and Sun Microsystems. Wood has significant financial interest in Sun Microsystems. The views expressed herein are not necessarily those of the NSF, Intel, or Sun Microsystems.

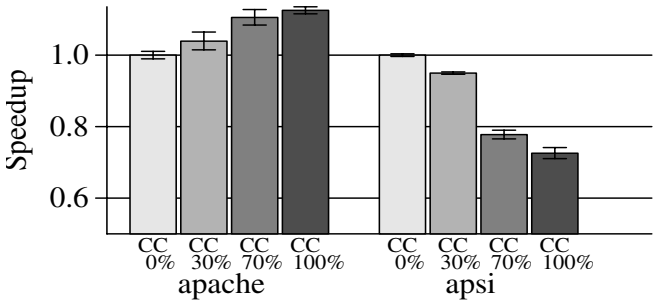

Figure 1: Cooperative Caching (CC) Speedup: 16 MB CMP configuration

- We propose ASR, the first hardware mechanism that dynamically controls replication. When applied to SPR, ASR provides a dynamically adaptive CMP cache hierarchy that improves performance by as much as 12% versus previous replication policies. Furthermore, ASR adds only 1.2% storage overhead to a future on-chip cache hierarchy.

Section 2 characterizes the CMP working sets of the 8 evaluated workloads, Section 3 describes ASR and Section 4 describes SPR and the ASR implementation. Section 5 details the evaluation methodology, Section 6 presents simulation results, and Section 7 summarizes related work.

2. Characterizing CMP Working Sets

To understand the potential benefits and costs of replication, this section analyzes the sharing 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. In particular, the study simulates an eight-processor CMP executing various commercial and scientific workloads under the Solaris 9 operating system. To mitigate cold start effects, all simulations run long enough that total L2 cache misses significantly outnumber the L2 block frames. Section 5 describes the simulation environment and workloads in more detail.

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

Table 1 shows—in general—that while many requests are to shared data, single-requestor blocks consume the majority of the cache capacity. Shared read-only blocks account for 42%-71% of requests for the four commercial workloads and two of the scientific workloads. Yet single-requestor blocks account for over 50% of L2 cache capacity for all workloads and over 90% for Jbb, Apsi, Barnes, and Ocean. In comparison, shared read-only and shared read-write data consume relatively little capacity, with the maximum being less than 30%.

Replicating shared blocks 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 shared read-only blocks in Apache, Jbb, Oltp, Zeus, and Art are requested by 3.0 to 4.5 processors, on average, during their on-chip cache lifetimes. Fully replicating these blocks could increase the effective working set by 25-74%.

Fortunately, shared read-only blocks exhibit strong locality, especially for commercial workloads. Figure 2a plots the cumulative request distribution versus the cumulative capacity distribution for shared read-only blocks. For all commercial workloads, the top 20% of blocks account for over 90% of requests and the top 3% of blocks account for over 70% of requests. Conversely, Figure 2b illustrates that shared read-write blocks have much less locality: the top 20% of blocks only account for 75% or less of requests. 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. For this reason, we focus on replicating shared read-only blocks in this paper.

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 3 illustrates this risk by plotting the normalized hit ratios for fully-associative caches up to 32 MB.

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

Table 1: L2 Cache Request and L2 Cache Capacity Profile

	Single Requestor		Shared Read-Only			Shared Read-Write		
Benchmark	% of Requests	% of Capacity	% of Requests	% of Capacity	Avg. # of Sharers	% of Requests	% of Capacity	Avg. # of Sharers
apache	11%	51%	44%	20%	3.7	44%	29%	2.8
jbb	57	91	42	10	3.5	1	< 1	2.4
oltp	4	51	71	21	4.5	25	28	3.6
zeus	20	71	54	11	3.0	25	18	2.3
apsi	> 99	> 99	< 1	< 1	7.3	< 1	< 1	2.8
art	53	71	46	29	3.0	< 1	< 1	2.3
barnes	19	93	74	4	3.2	7	3	2.1
ocean	94	98	1	< 1	4.7	5	1	2.1

0-7695-2732-9/06 $20.00

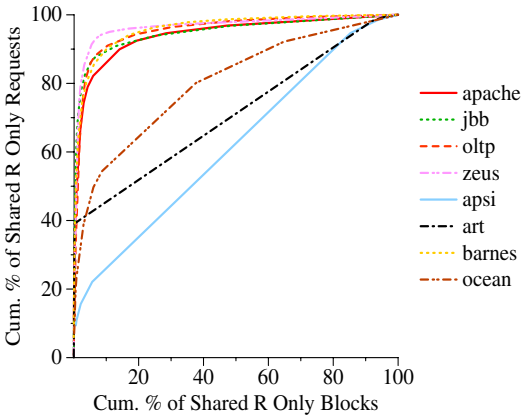

Figure 2: a) Request to Block Distribution: Shared Read-Only

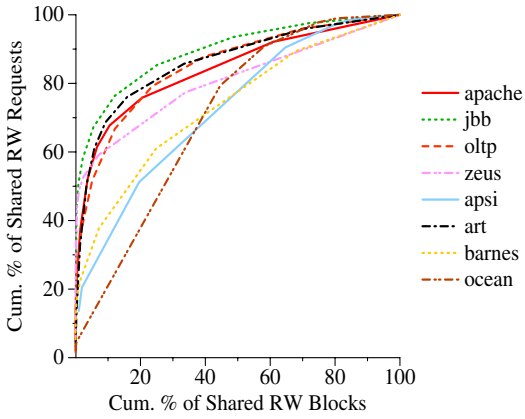

Figure 2: b) Request to Block Distribution: Shared Read-Write

This graph demonstrates the sensitivity that many workloads have to small changes in cache size. For example, Ocean and Art have critical working set sizes of 4 MB and 8 MB, respectively. Decreasing the effective cache capacity below these thresholds has a dramatic negative impact on performance. All of the scientific workloads exhibit clearly identifiable working set boundaries, while the commercial workloads have less pronounced transitions. Ideally, a replication policy would avoid decreasing capacity below these critical thresholds.

3. ASR: Adaptive Selective Replication

ASR seeks the optimum replication level by balancing the benefits of replication against the costs. Section 3.1 introduces the simple memory system performance model underlying ASR and Section 3.2 describes ASR's replication algorithm.

3.1 Replication and CMP Cache Performance

To the first order, L2 cache block replication improves memory system performance when it reduces the average L1 miss latency. The following equation describes the average cycles for L1 cache misses normalized by instructions executed:

$$\frac{\text{L1 miss cycles}}{\text{Instruction}} = \frac{(P_{localL2} \times L_{localL2})}{(Instructions/L1misses)} + \frac{(P_{remoteL2} \times L_{remoteL2})}{(Instructions/L1misses)} + \frac{(P_{miss} \times L_{miss})}{(Instructions/L1misses)}$$

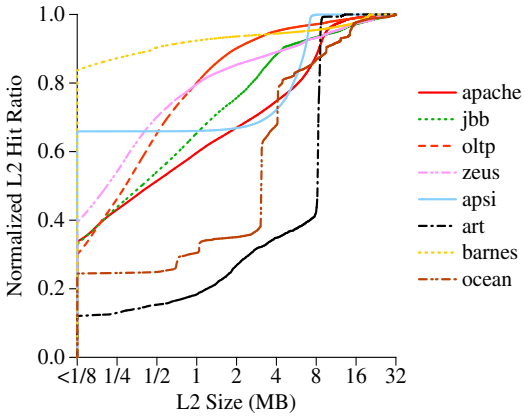

Figure 3: Normalize L2 Cache Hit Ratios

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 *localL2* and *remoteL2* terms represent the memory cycles spent on L2 cache hits and the third term depicts the memory cycles spent on L2 cache misses. Replication increases the probability that L1 misses 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 is a reduction in 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.

Optimal performance often arises from an intermediate replication level. Figure 4 graphically depicts this tradeoff. The *Replication Benefit* curve, Figure 4a, illustrates the trend that increasing replication reduces L2 cache hit cycles. Due to the strong locality of shared read-only requests, a small degree of L2 replication can significantly reduce L2 hit cycles by moving many previous remote L2 hits into the local cache. In contrast, increased replication gradually reduces L2 hit cycles because fewer unique blocks on-chip lead to fewer total L2 hits. The *Replication Cost* curve, Figure 4b, illustrates that increasing L2 replication increases the memory cycles spent on off-chip misses. The *Replication Effectiveness* curve, Figure 4c, combines the benefit and cost curves and plots the total memory cycles. Because the benefit and cost curves are generally convex and have opposite slopes, the minimum of the Replication Effectiveness curve often lies between allowing all replications and no replications. ASR estimates the slopes of the benefit and cost curves to approximate the optimal replication level.

3.2 Balancing Replication via ASR

By dynamically monitoring the benefit and cost of replication, ASR attempts to achieve the optimal level of replication. ASR identifies discrete replication levels and makes a piecewise approximation of the memory cycle slope. Thus ASR

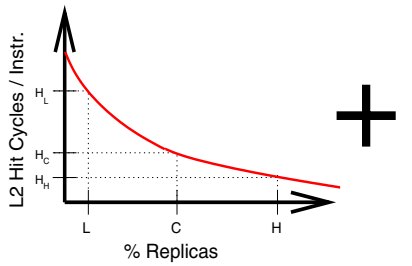

Figure 4: a) Replication Benefit

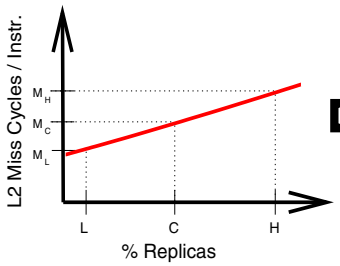

Figure 4: b) Replication Cost

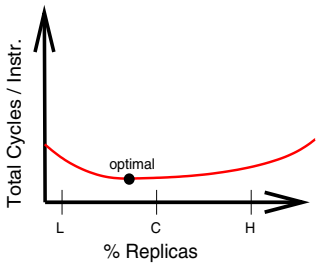

Figure 4: c) Replication Effectiveness

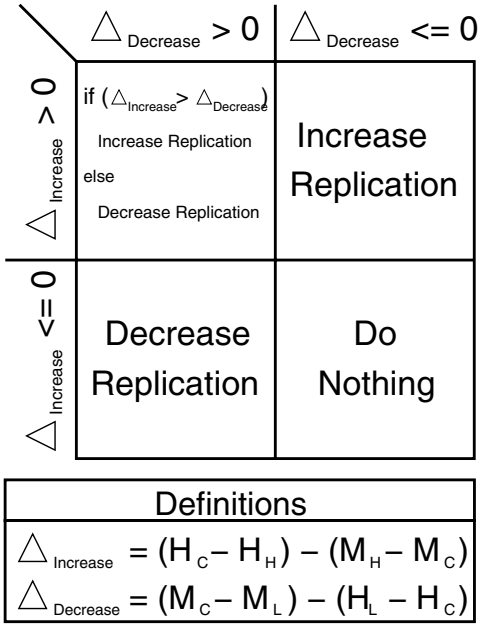

Figure 5: ASR Decision Table for Adjusting Replication

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: (i) increasing replication to the next higher level, labeled H, (ii) decreasing replication to the next lower level, labeled L, or (iii) leaving the replication 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 benefit of increasing replication *(*decrease in L2 hit cycles, H_C - H_H*)*; (2) the cost of increasing replication *(*increase in L2 miss cycles, M_H - M_C*)*; (3) the benefit of decreasing replication, *(*decrease in L2 miss cycles, M_C - M_L*)*; and (4) the cost of decreasing replication *(*increase in L2 hit cycles, H_L - H_C*)*.

By comparing these cost and benefit counters, ASR will increase, decrease, or leave unchanged the replication level. Figure 5 presents ASR's decision table for adjusting replication. $\Delta_{Increase}$ and $\Delta_{Decrease}$ summarize the cost and benefit counters: positive values indicate that increasing or decreasing replication, respectively, will improve performance. When both $\Delta_{Increase}$ and $\Delta_{Decrease}$ are positive, ASR chooses the direction with the greater predicted benefit.

Table 2: SPR Replication Levels

Level	0	1	2	3	4	5
Probability	0	1/64	1/16	1/4	1/2	1
Threshold	0	4	16	64	128	256

4. Implementing ASR using SPR

Implementing ASR requires a CMP cache framework that supports multiple replication levels. Cooperative Caching [8] is one possibility, but this scheme requires an expensive central tag structure. Section 4.1 introduces the simpler Selective Probabilistic Replication (SPR) design which uses distributed state to make local replication decisions. Section 4.2 describes the additional hardware needed to implement ASR, with Beckmann's dissertation [5] providing further detail. Finally, Section 4.3 summarizes ASR's storage and energy overhead.

4.1 SPR: Selective Probabilistic Replication

Like most earlier replication proposals, SPR assumes private L2 caches and selectively limits replication on L1 evictions. SPR uses a non-inclusive Token Coherence broadcast protocol [22] and ring writebacks [30] to eliminate the need for a central tag structure (like Cooperative Caching) or a designated home node (like Victim Replication). While token coherence simplified SPR's implementation, SPR is not dependent on token coherence and instead could have used a non-home-node directory protocol, e.g., AMD's HyperTransport cache coherence protocol [2]. On an L1 cache eviction, SPR writes a shared block back to its local L2 if (i) the block was already allocated in the local L2 or (ii) the replication policy (below) allocates a new block. Otherwise, SPR uses a ring writeback to merge the block with an existing remote L2 copy. Specifically, L1 writeback messages are passed clockwise between private L2 caches to search for an already allocated copy or an empty L2 block.

To avoid extra delay on writes due to coherence invalidations, SPR only replicates shared read-only data. To identify which cache blocks are shared and read-only, SPR uses the per-block dirty bit in combination with an extra per-block shared bit. The L1 and L2 cache tags set the shared bit when receiving a request from a processor different than the current sharer. Similar to the dirty bit, once the shared bit set, it is not reset until the block is replaced. When the dirty bit is not set and shared bit is set, the block is considered shared read-only.

On L1 cache writebacks, SPR uses probabilistic filtering to decide when to replicate a block. To simplify the replication

0-7695-2732-9/06 $20.00

process, SPR supports six discrete replication levels (Table 2). 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. When an L1 cache evicts a shared read-only block and the block is not found in the local L2 cache, the replication probability determines whether to replicate the block locally. Specifically, a linear feedback shift register [13] generates an 6-bit pseudo-random number which it compares to the current replication threshold (i.e., if random < threshold, then replicate). Like all SPR logic, the pseudo-random number generator does not impact L2 cache access latency and is accessed only on replacement. The probabilistic policy biases replications to frequently requested blocks because frequently requested L2 blocks are also those blocks frequently evicted from the L1 caches.

4.2 Implementing ASR

Determining whether to increase or decrease replication requires knowing whether a block would be replicated at the next higher or next lower level. ASR identifies these blocks by comparing the random number not just against the current replication threshold, but also against the thresholds for the next higher and lower levels. Note that because the thresholds are monotonic, all decisions to replicate a block at level i will also be made at level i+1. ASR uses the information about whether a block should be replicated at the current, next lower, or next higher levels to maintain the mechanisms described below.

ASR uses four separate mechanisms to estimate the costs and benefits of replication and another mechanism to trigger a replication analysis that could change the replication level.

Benefit of Increasing Replication (H_C - H_H). To determine the benefit of increasing replication, ASR identifies the blocks not replicated at the current replication level, but that would have been replicated with the next higher level. Specifically, each processor stores 8-bit partial tags [16] of the blocks that would have been replicated with the next higher replication level using separate 16 K entry, 16-way set-associative buffers called the Next Level Hit Buffers (NLHBs). When a request hits in a remote L2 cache, the local NLHB is checked to determine if the request could have been a local hit if replication was increased. If so, ASR increments its (H_C - H_H) counter by the number of cycles that would be saved by a local L2 hit versus a remote L2 hit.

Cost of Increasing Replication (M_H - M_C). ASR estimates the cost of increasing the replication level by estimating the utilization of soon-to-be-evicted L2 cache blocks. In other words, these are the unique L2 blocks that would exist off-chip if replication was increased. Specifically, ASR monitors the last 1 K of least recently used L2 blocks. A monitor size greater than 1 K provides little additional benefit due to the low locality of these blocks. Because precisely determining the recently used cache blocks is prohibitively expensive in hardware, ASR uses way and set counters [32] to estimate which blocks are 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 [28] to estimate the LRU position of the requested set-group. If a request hits an L2 block not identified as a current replica and the block's way and set-group position lies within the last 1 K of LRU blocks, the (M_H -M_C) counter is incremented by the off-chip memory latency.

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 VTB stores 16-bit partial tags of the most recently evicted blocks in a 1 K entry 16-way set associative buffer. The VTB only stores tags that were evicted due to the current replication level, but would not have been evicted with the next lower level. When a replication associated with the current replication level causes an L2 eviction, the VTB allocates the evicted tag. The VTB stores other L2 eviction tags only 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 SPR replication level decreases, ASR clears the VTB because the tags currently stored no longer correspond to the new lower replication level.

Cost of Decreasing Replication (H_L - H_C). To estimate the cost of decreasing replication, ASR identifies blocks that are replicated at the current replication level, but would not be replicated at the next lower level. Specifically, 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 its (H_L - H_C) counter by the difference between a remote L2 hit and a local L2 hit. When the SPR replication level increases, ASR clears the current replication bits because the bits no longer correspond to the new replication level.

Triggering a Cost-Benefit Analysis. Like all adaptive systems, ASR should respond quickly, but not too quickly, to changes in workload behavior. ASR does this using a two-step process. First, ASR waits until it observes enough events to ensure a fair cost/benefit comparison. Specifically, ASR triggers an evaluation when the number of local L2 replications or NLHB allocations exceed the 1 K entry monitor size. Thus, the time interval between replication evaluations is not fixed, nor do the evaluations require chip-wide coordination. Rather, the evaluation intervals depend only on the frequency of local replication opportunities. Upon triggering an evaluation, ASR performs the comparison described in Section 3.2 to determine if and how the replication level should be changed. Second, ASR provides hysteresis by waiting until four consecutive evaluation intervals predict the same change before making an actual change to the replication level. After each evaluation, all four counters are cleared.

4.3 Storage and Energy

ASR adds a small storage overhead to the on-chip cache hierarchy and should have minimal impact on energy consumption. For an eight processor CMP, Table 3 breaks down ASR's storage requirement for two cache configurations: a 4 MB aggregate L2 cache with 16 KB L1 caches and a 16 MB aggregate L2 with 64 KB L1s. Table 3 demonstrates that ASR scales well to bigger caches because many of its structures are cache size independent. For instance, between the 4 MB and 16 MB configurations, ASR's storage overhead only grows by 40 KB. ASR's size is mostly independent of cache size because it only monitors the marginal benefits and costs of replication, instead of monitoring replication's effectiveness across the entire cache. Later, Section 6.2 directly compares ASR's storage overhead with the previous proposals [8, 9, 41].

0-7695-2732-9/06 $20.00

Table 3: ASR Storage Overhead

Overhead	Bits	K Entries		K Bytes	
		4 MB CMP	16 MB CMP	4 MB CMP	16 MB CMP
per L1 block	1	4	16	0.5	2
per L2 block	2	64	256	16	64
NLHBs	8	128	128	128	128
VTBs	16	8	8	16	16
Total KBytes (including counters)				161.5	211
% increase of on-chip cache capacity				3.7%	1.2%
Area (90 nm & 45 nm tech. respectively)				~ 3 mm^2	~ 1 mm^2

While ASR costs some bits, it doesn't consume energy for passing messages between processors for coordinating replication level changes. Each L2 cache makes a local replication decision. SPR's replication logic lies on the non-latency critical L1 replacement decision and is a simple probabilistic choice. Additionally, ASR's tables and counters are also non-latency critical and are only accessed on L1 and L2 misses. Therefore, ASR's logic will be accessed relatively infrequently and can use high-Vt low-leakage transistors [25]. Also, ASR's cost-benefit model could be extended to account for the dynamic power consumed by local versus remote L2 hits. We leave this for future work.

5. Methodology

By using full-system simulation based on Simics [36] and the GEMS toolset [37], we evaluate ASR against alternative cache designs. This section describes the alternative caches, system parameters, and workloads that we use in our simulation study.

5.1 Alternative Cache Designs

Section 6 compares ASR against two baseline configurations—shared L2 and private L2 caches—and the previous replication proposals: Victim Replication [41], CMP-NuRapid [9], and Cooperative Caching [8].

CMP-Shared. As illustrated in Figure 6a, the CMP-Shared design assumes a Non-Uniform Cache Architecture (NUCA) [17]. CMP-Shared statically maps the addresses across all on-chip L2 banks, thus forming a shared cache with non-uniform latency. On an L1 miss, a processor sends its request to the appropriate L2 bank which may forward the request to L1 sharers or memory. By disallowing L2 replication, the CMP-Shared achieves the best capacity, but by not exploiting the distance locality between processors and L2 banks, it incurs the highest access latency.

CMP-Private. The CMP-Private design (Figure 6b) assigns each cache bank private to a processor. Similar to the Itanium 2 microprocessor [24], 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. Unlike other baseline private cache designs [8], CMP-Private allows direct cache-to-cache transfers of clean data. L1 misses and replacements are directed to the local private L2 bank and other processors cannot allocate into a remote bank. Thus, CMP-Private migrates [6, 14] single requestor data and replicates all shared data without the storage overhead of home blocks associated with a distributed

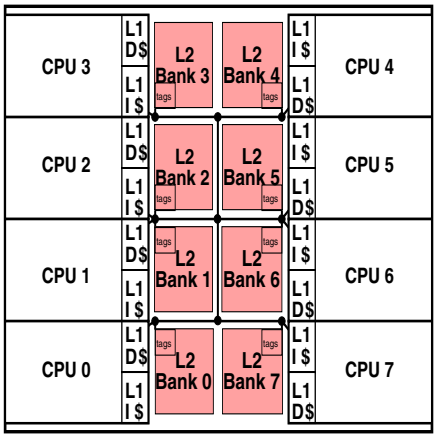

Figure 6: a) Layout of CMP-Shared

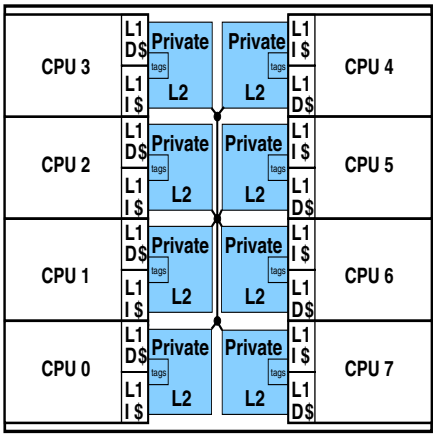

Figure 6: b) Layout of CMP-Private

directory protocol. However, CMP-Private's unrestricted replication of shared data can increase off-chip misses and coherence invalidations.

In addition to supporting ASR, SPR's selective replication framework can support previously proposed replication policies with relatively simple changes.

SPR-VR. Victim Replication [41] targets an on-chip directory protocol and statically assigns blocks to home nodes (like CMP-Shared). Non-home nodes replicate blocks locally, except when a L2 cache set is already full of home blocks that have remote sharers. Using a random replacement policy, non-shared home blocks are evicted before replicas. SPR-VR implements Victim Replication's replication policy by adding 1-bit per L2 cache block to identify replicas and by disallowing replications when the local cache set is filled with owner blocks with identified sharers. Victim Replication's distributed directory protocol wastes significant storage by forcing home nodes to store cache blocks regardless if the home node actually used the block. Thus, replicating shared data overlapped with migrating single requestor data away from its home bank. Although it requires more bandwidth, SPR-VR should perform strictly better than the original Victim Replication implementation because the token broadcast protocol [22] removes the home node storage overhead.

0-7695-2732-9/06 $20.00

Table 4: Comparison of Configuration Parameters

Parameters	Current CMP	Future CMP
processor model / cycle time	in-order / 1.4 GHz	out-of-order / 5.0 GHz
split L1 I & D caches	16 KB each, 4-way, 2 cycles	64 KB each, 4-way, 3 cycles
aggregate L2 cache sizes	4 MB 16-way pseudoLRU [28]	16 MB 16-way pseudoLRU [28]
avg. shared L2 / local L2 / remote L2 latency	25 / 12 / 34 cycles	44 / 20 / 50 cycles
memory latency	150 cycles	500 cycles
memory bandwidth	28 GB/s	50 GB/s

SPR-NR. CMP-NuRapid [9] maintains coherence using a shared bus and per-processor decoupled tag arrays with indirect data block pointers (6% overhead). CMP-NuRapid's replication policy allocates a local L2 tag after the first request and then locally allocates the actual L2 data block upon a second request. SPR-NR removes the shared bus overhead and incorporates CMP-NuRapid's replication policy by storing a 1-bit counter per remote processor for each L2 block (1.4% overhead). A processor's first request sets its associated bit; a second request by the same processor causes local allocation of the L2 block.

SPR-CC. Cooperative Caching (CC) [8] uses a centralized duplicate tag structure to identify singlets, i.e., an L2 block with one on-chip copy, and biases replacements to evict non-singlets first. Cooperative Caching retains active singlets by spilling them to a remote L2 cache. SPR-CC models the centralized tag structure using an idealized distributed tag structure.

5.2 System Parameters

Our evaluation studies two different SPARC V9 8-processor CMP configurations targeting current and future technology. The Sun Niagara [18, 20] inspired the first CMP configuration, (the second column of Table 4), and the second configuration presents a CMP assuming 2010 technology [12] (column 3 of Table 4). The 15-stage out-of-order (OoO) cores of the Future CMP design are 4-wide superscalar processors using 64-entry instruction scheduling windows with 128-entry ROBs. Each OoO core predicts branches using a 3.5 KB YAGS direct branch predictor, a 256-entry cascaded indirect branch predictor, and a 64-entry return address stack predictor. Table 4 also includes the average load-to-use latencies [1] for the shared cache, as well as, the local and remote banks in the private cache. Both configurations utilize 64-byte cache blocks and 4 GB of DRAM.

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 inclusive with the L1 caches and maintains up-to-date L1 sharer knowledge. All evaluated designs also incorporate strided prefetchers between the L1 and L2 caches, as well between the L2 caches and memory. The prefetcher is based on the IBM Power 4 [34], except it issues prefetches for both load and stores.

The intra-chip and inter-chip networks are modeled in detail, including all messages required to implement the coherence protocol. The on-chip links are 64-bytes wide and the off-chip bandwidth is specified in Table 4. Virtual cut-through routing is used with three 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.

Table 5: Evaluation Methodology

Benchmark	Fast Fwd.	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
barnes	NA	1.9	run completion
ocean	NA	2.4	run completion

5.3 Workloads

We studied the CMP cache designs using commercial and scientific workloads. Alameldeen *et al.* [3] described the first three commercial workloads and Xu *et al.* described Zeus [39]. We also studied four scientific workloads: two SPLASH2 benchmarks [38] Barnes (128 k-particle) and Ocean (514×514), and two SpecOMP benchmarks [4]: Apsi and Art. We used a work-related throughput metric to address multithreaded workload variability [3]. 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 5. 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 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 95% confidence intervals.

6. Evaluation

6.1 Replication Capacity and Memory Cycles

The optimal replication point shifts depending on workload behavior and CMP configuration. Figure 7 displays the L2 hit cycles-per-instruction, L2 miss cycles-per-instruction, and the Total cycles-per-instruction curves for both CMP configurations. Each point on the curve corresponds to a static SPR replication level.

For Current CMP, 6 of 8 workloads prefer either minimum or maximum replication, while Apache and Oltp prefer intermediate replication. The first row of graphs (Figure 7a-c) presents the results for the Current CMP configuration with a 4 MB aggregate

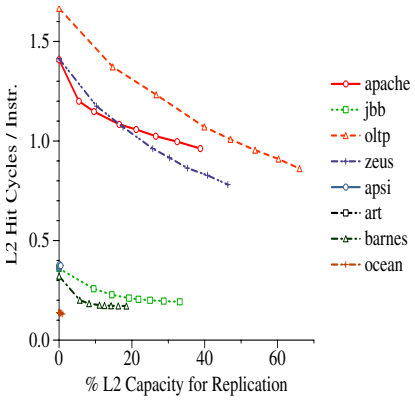

Figure 7: a) L2 Hit Cycles / Instr. (Current CMP)

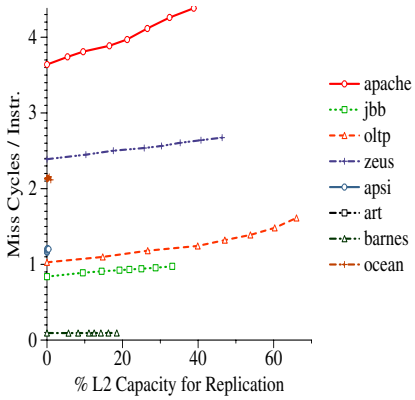

Figure 7: b) L2 Miss Cycles / Instr. (Current CMP)

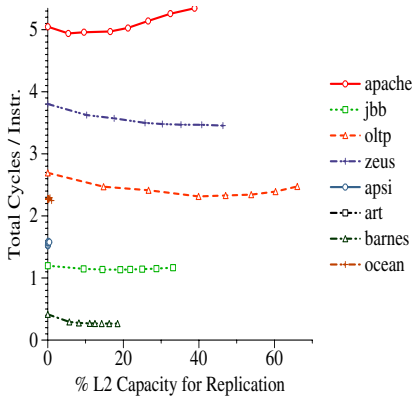

Figure 7: c) Total Cycles / Instr. (Current CMP)

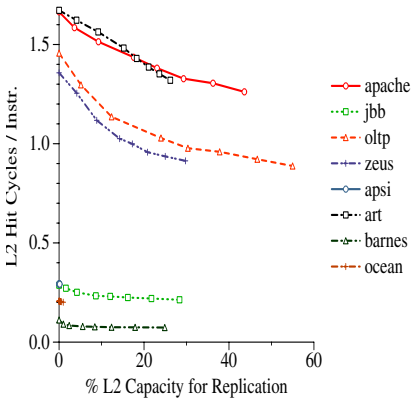

Figure 7: d) L2 Hit Cycles / Instr. (Future CMP)

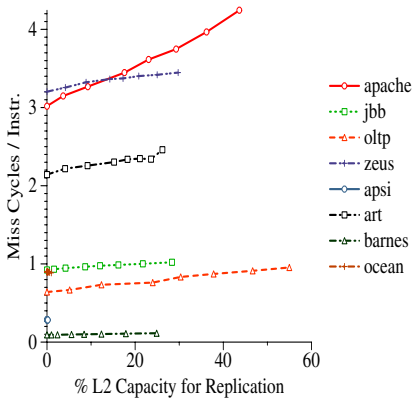

Figure 7: e) L2 Miss Cycles / Instr. (Future CMP)

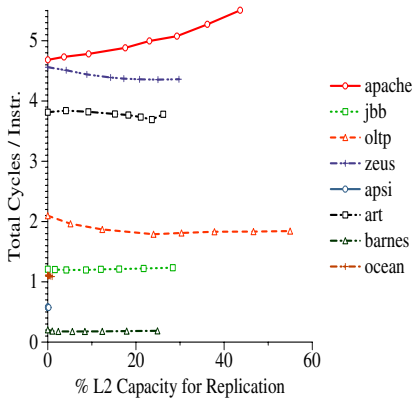

Figure 7: f) Total Cycles / Instr. (Future CMP)

L2 cache capacity. The L2 hit cycles-per-instruction curves for the workloads: Apache, Jbb, Oltp, Zeus, and Barnes (Figure 7a) demonstrate how selective replication can exploit the request locality of shared read-only data. The slopes of these five convex curves show that limited replication attains most of the latency reduction possible with unlimited replication. For instance in Apache, devoting 10% of L2 capacity to replication reduces L2 hit cycles-per-instruction by 0.3, but allowing replicas to consume 30% more capacity provides less than a 0.2 additional reduction. In contrast, Figure 7b illustrates the L2 miss cycles-per-instruction curves have a more consistent slope. For example, Apache's miss cycles-per-instruction curve roughly increases by 0.1 for every 5% increase in replication capacity. The resulting total cycles-per-instruction curves (Figure 7c) reveal the optimal point of replication for each workload using the Current CMP configuration. Replication has little effect on the scientific workloads Apsi, Art, and Barnes, while the workloads Jbb, Zeus, and Barnes prefer maximum replication. The most interesting cases, Apache and Oltp, prefer a replication capacity between the minimum and maximum.

For the Future CMP configuration, the second row of memory curves (Figure 7d-f) show that the optimal level of replication shifts towards less replication. For most workloads, the normalized L2 hit cycle curves (Figure 7d) maintain the same basic shape as those of Figure 7a. However, Art demonstrates how balancing replication becomes more important with larger caches [15] because its 8 MB working set (Figure 3) now fits in Future CMP's larger cache. Figure 7e illustrates Future CMP's slower memory latency causes the L2 miss cycle slopes to substantially increase with respect to those in Figure 7b. The result is the miss cycle curves have a greater impact on the total cycle curves. For instance the optimal replication level for Apache and Oltp shifted from 5% and 40%, respectively, for the Current CMP configuration, to 0% and 20% for the Future CMP configuration.

6.2 Adapting to Workload Behavior

By dynamically monitoring the changes in L2 hit and miss cycles, ASR matches the level of replication within each private L2 cache to the behavior of each individual processor. Figure 8 illustrates SPR-ASR's dynamic adjustment of each private L2 cache's replication level over time. These results use the Future CMP configuration, each processor's replication level is initialized to level 4, and each point represents when SPR-ASR evaluated of its counters.

Figure 8a shows that the Apache workload benefits from SPR-ASR's adaptability. For the first 200 million cycles the replication level fluctuates around 4, then drops to hover around levels 2 and 3 for the remainder of the execution, with a few

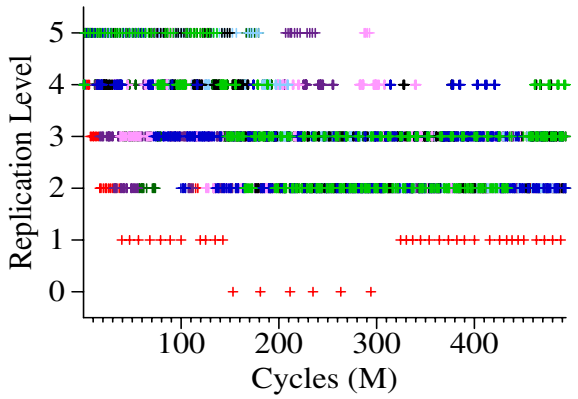

Figure 8: a) Future CMP ASR Adaptability: Apache

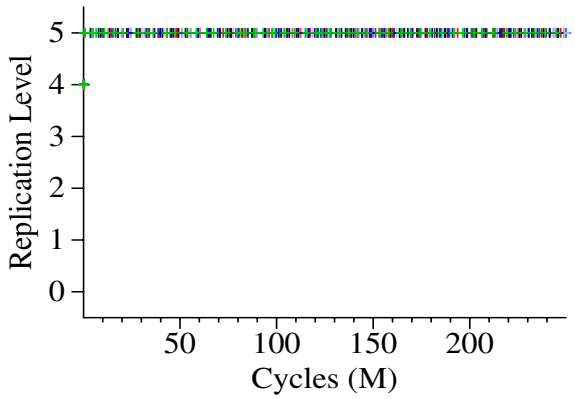

Figure 8: b) Future CMP ASR Adaptability: Oltp

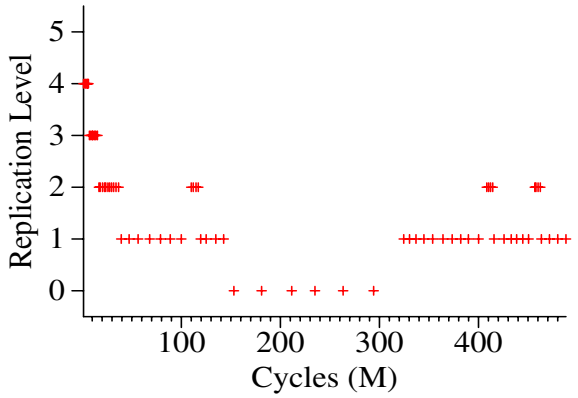

Figure 9: a) Future CMP: ASR Adaptability: Apache (Processor 0)

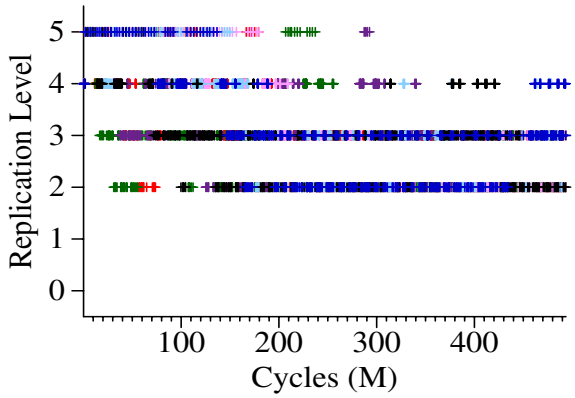

Figure 9: b) Future CMP: ASR Adaptability: Apache (Processors 1-7)

outliers either way. At this relatively low replication level, only 5% of the L2 cache contains replicas on average. Interestingly, Figure 9 shows that the low outliers are entirely due to processor 0, the one processor that executes almost exclusively OS code. Apparently, the OS's large working set size largely eliminates the benefit of replication. Thus Apache demonstrates the benefit of maintaining SPR-ASR's replication level on a per-processor basis.

Conversely, a static, global replication level would suffice for the Oltp workload. As illustrated in Figure 8b, SPR-ASR quickly adjusts all processors to level 5, maximizing replication. The result is that replicas account for 52% of the L2 capacity on average.

As Figure 8a shows, SPR-ASR can require well over a 100 million cycles to reach steady state, requiring several hours of simulation. To reduce simulation time, the remainder of this section initializes the replication levels to their steady-state value.

6.3 Comparison of Replication Schemes

Performance. For workloads where replication interferes with the active working set, SPR-ASR outperforms the alternative CMP cache designs. For workloads where replication capacity has little effect, SPR-ASR performs at least as well as the other designs. Figure 10 shows the normalized runtime of each CMP design executing the eight workloads. The two SPR-CC bars represent the best- and worst-performing Cooperative Caching percentages (where 100% maximizes replication and 0% minimizes it). For all workloads except Oltp, the private cache designs (excluding worst SPR-CC) exploit the relatively fast memory latency of the Current CMP configuration and improve performance by 0-29% versus CMP-Shared. For Oltp, SPR-ASR and best SPR-CC perform equivalently to CMP-Shared, while the other private cache designs degrade performance by 3-7%. SPR-ASR limits replication to 35% of L2 capacity, resulting in a 43% increase in off-chip cycles versus CMP-Shared. In contrast, the other private CMP designs devote as much as 64% of L2 capacity to replicas causing up to a 96% increase in off-chip cycles versus CMP-Shared. Overall, the best performing SPR-CC percentage achieves similar performance to that of SPR-ASR. However, the best SPR-CC percentage varies by workload—Apache and Oltp do best at 70%, Zeus at 30%, and the others at 0%—illustrating the benefit of an adaptive policy.

To provide further insight, Figure 11 shows the memory system cycle breakdown for the Apache and Oltp workloads to indicate where the time is spent in the memory system. The 'Local L1' and 'Local/Shrd L2' segments display the fraction of the average memory access time contributed by local L1 and L2 hits respectively (for CMP-Shared 'Local/Shrd L2' indicates shared 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 indicates the cycles spent on off-chip misses.

As forecast by the total cycles-per-instruction curve (Figure 7c) the four private cache designs that restrict replication (SPR-VR, SPR-NR, SPR-CC, and SPR-ASR) attain better performance for Apache than CMP-Private, which allows all

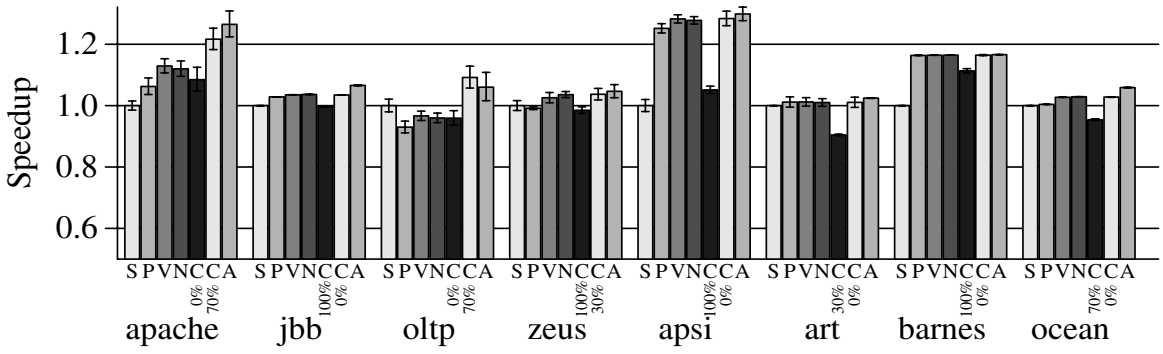

Figure 10: Current CMP Speedups
(S: CMP-Shared, P: CMP-Private, V: SPR-VR, N: SPR-NR, C: SPR-CC, A: SPR-ASR)

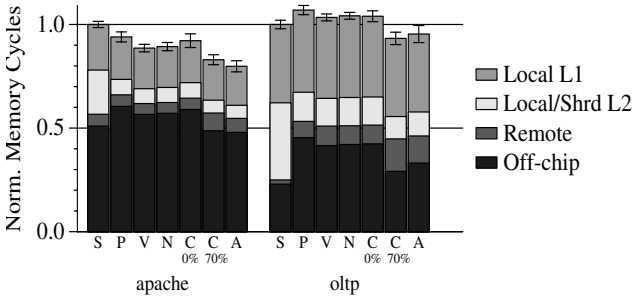

Figure 11: Current CMP Memory Cycles
(S: CMP-Shared, P: CMP-Private, V: SPR-VR, N: SPR-NR, C: SPR-CC, A: SPR-ASR)

replication. Specifically, SPR-ASR achieves the greatest performance improvement (24% and 19% versus CMP-Shared and CMP-Private respectively) by restricting replication to only 5% of L2 capacity, while SPR-VR and SPR-NR allow replicas to consume more than 38% of capacity. Figure 11 shows the limited replication capacity enforced by SPR-ASR exploits the strong locality of shared read-only requests. Specifically, SPR-ASR achieves almost as many local L2 hits as SPR-NR (82%), while SPR-ASR's greater effective cache capacity reduces off-chip miss cycles by 20%.

SPR-ASR also improves the Future CMP configuration despite the fact that higher memory latency and out-of-order processors [26] change the performance tradeoff between shared and private caches. For Apache, Apsi, and Barnes, the performance advantage private cache designs exhibit over CMP-Shared diminishes by 12-23%, while the private cache performance advantage increases for Oltp and Ocean by 11-15%. Figure 13 breaks down the memory cycles for Apache and Oltp. Because Oltp's working set better fits in Future CMP's larger cache, the private cache organizations utilize replication to exploit the faster private L2 caches. Oltp's frequently requested shared read-only working set (Table 1) especially benefits from replication, enabling the private cache designs to improve performance by 6-12% versus CMP-Shared. However, Apache's larger working set (Figure 3) exposes Future CMP's slower memory latency and all private cache designs except the best SPR-CC and SPR-ASR suffer substantial performance degradation versus CMP-Shared. Specifically, CMP-Private, SPR-VR, and SPR-NR suffer a performance degradation of 7-13% versus CMP-Shared, while SPR-ASR achieves the greatest performance improvement versus CMP-Shared—5%. Though SPR-CC with 100% performs similarly to SPR-ASR in Apache, SPR-CC with 0% suffers nearly the same performance degradation as SPR-VR and SPR-NR. Conversely, SPR-ASR does not require external tuning and dynamically identifies replication's lack of benefit. The result is SPR-ASR increases off-chip misses by only 13% and improves performance by 2% versus CMP-Shared, 17% versus CMP-Private, and 12% versus SPR-VR, and SPR-NR.

Performance Summary. Overall, SPR-ASR significantly improves performance for workloads where shared read-only replication conflicts with the active working set, e.g. Apache and Oltp. For other workloads, SPR-ASR always performs competitively, if not better than the best alternative. SPR-ASR's performance stability ensures CMP caches will provide good performance to a wider variety of workloads.

Storage Overhead. SPR-ASR achieves better performance for less storage overhead than the previous hybrid cache designs because it relies on SPR's probabilistic filtering rather than a more hardware intensive replication mechanism, such as those used by CMP-NuRapid and Cooperative Caching. Instead, SPR-ASR targets its storage overhead to dynamically monitoring replication's cost and benefit. For the Current and Future CMP configurations, Table 6 compares the storage overhead of SPR-VR, SPR-NR, SPR-CC, and SPR-ASR. SPR-VR's and SPR-ASR's replication mechanisms add only one bit per L2 cache block for respectively identifying replica and shared blocks. Thus, these mechanisms scale well with increasing the aggregate L2 cache size. In comparison, CMP-NuRapid's SPR implementation adds 7-bits per L2 block—1-bit counter for each remote L2 cache—and Cooperative Caching's SPR implementation requires a duplicate tag per L2 cache block.

7. Related Work

7.1 Multiprocessor Memories

A large body of previous work exists in studying data replication in the context of flat multiprocessors [10]. Specifically, throughout the previous decade significant work has compared hardware solutions such as CC-NUMA and Flat COMA architectures [31, 42], along with software [7, 35] and hybrid hardware/software combinations [11, 29]. The Flat COMA protocol [29, 31] removed the slow ordered network of hierarchical COMA machines allowing data to migrate and

0-7695-2732-9/06 $20.00

Table 6: Storage Overhead Comparison

SPR Cache Design	Replication Mechanism		Adaptive Mechanism	
	Current CMP	Future CMP	Current CMP	Future CMP
Victim Replication	8 KB	32 KB	Not Applicable	Not Applicable
CMP-NuRapid	56 KB	224 KB	Not Applicable	Not Applicable
Cooperative Caching	255 KB	886 KB	Not Applicable	Not Applicable
ASR	8.5 KB	34 KB	153 KB	177 KB

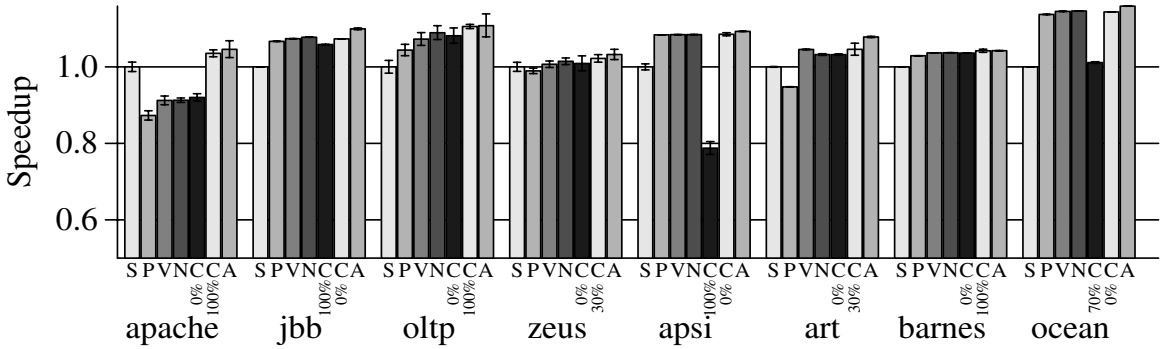

Figure 12: Future CMP Speedups
(S: CMP-Shared, P: CMP-Private, V: SPR-VR, N: SPR-NR, C: SPR-CC, A: SPR-ASR)

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.* [35] proposed an OS mechanism that adapted the number of pages migrated and replicated to a processor's local memory. Zhang *et al.* [42] studied the working sets of various workloads running on a NUMA system. Compared 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.* [14] investigated sharing in a CMP-NUCA cache and concluded allowing some replication between cache banks was advantageous. Liu *et al.* [21] 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, ASR dynamically analyzes workload behavior and adapts replication on a per block basis to match the current demand.

The most closely related proposals to Adaptive Selective Replication are the previously discussed Victim Replication [41], CMP-NuRapid [9], and Cooperative Caching [8] proposals. All three designs reduce replica blocks, but their static mechanisms tend to favor certain workloads and do not dynamically adjust to changes in workload behavior and system constraints. Cooperative Caching does introduce using probability to control replication, but does not propose a mechanism to actually adjust the probability. Through slight modification, ASR monitoring hardware could provide such a mechanism. Specifically, ASR's NLHB could be modified to determine the cost of the evicted replica blocks and ASR's VTB could be modified to determine the cost of the evicted singlet blocks. By comparing these costs to the estimated benefits of storing the current singlet and replica blocks, one could design an adaptive Cooperative Caching algorithm.

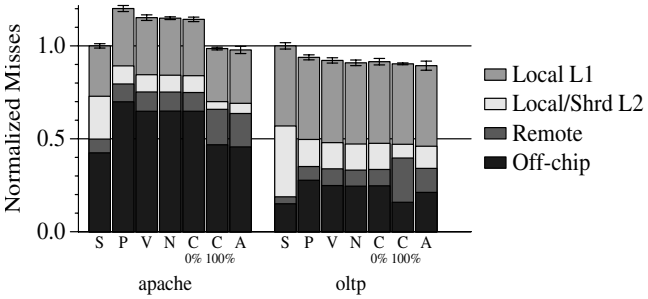

Figure 13: Future CMP Memory Cycles
(S: CMP-Shared, P: CMP-Private, V: SPR-VR,
N: SPR-NR, C: SPR-CC, A: SPR-ASR)

Finally, similar to ASR, Suh *et al.* [33] used set and way counters to monitor cache block utilization and Zhang *et al.* [40] used a miss tag buffer to track what cache misses could have been hits if the cache was full sized. However, the differences are Suh *et al.* used the monitoring information to dynamically partitioned ways in a set-associative cache among multiple threads and Zhang *et al.*'s miss tag buffer stored full size tags and was used to save energy in a automatically resizable cache.

8. Conclusions

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 uncontrolled 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. We propose Adaptive Selective Replication, which dynamically adapts

shared read-only data replication to exploit the latency advantage of private caches without wasting cache capacity due to excessive replication. By performing an opportunity analysis, ASR adjusts the degree of replication to match the current workload behavior and system configuration. We showed ASR performs at least as well as the best alternative design and improves performance for commercial workloads with large working sets.

Acknowledgements

We thank Luke Yen, Dan Gibson, 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. Ahmed, P. Conway, B. Hughes, and F. Weber. AMD Opteron Shared Memory MP Systems. In *Proc. of the 14th HotChips Symposium*, Aug. 2002.

[3] A. R. Alameldeen, *et al.* Simulating a $2M Commercial Server on a $2K PC. *IEEE Computer*, 36(2):50–57, 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] B. M. Beckmann. *Managing Wire Delay in Chip Multiprocessor Caches*. PhD thesis, University of Wisconsin, 2006.

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

[7] R. Chandra, S. Devine, B. Verghese, A. Gupta, and M. Rosenblum. Scheduling and Page Migration for Multiprocessor Compute Servers. In *Proc. of ASPLOS-6,* Oct. 1994.

[8] J. Chang and G. S. Sohi. Cooperative Caching for Chip Multiprocessors. In *Proceedings of the 33nd Annual International Symposium on Computer Architecture*, June 2006.

[9] 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.

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

[11] 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.

[12] I. T. R. for Semiconductors. ITRS 2005 Edition. Semiconductor Industry Association, 2005. http://www.itrs.net/Common/2005ITRS/Home2005.htm.

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

[14] 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.

[15] A. Jaleel, M. Mattina, and B. Jacob. Last Level Cache (LLC) Performance of Data Mining Workloads On a CMP: A Case Study of Parallel Bioinformatics Workloads. In *Proceedings of the 12th IEEE Symposium on High-Performance Computer Architecture*, Feb. 2006.

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

[17] C. Kim, D. Burger, and S. W. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches. In *Proc. of ASPLOS-10*, Oct. 2002.

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

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

[20] J. Laudon. Performance/Watt: The New Server Focus. In *Workshop on Design, Architecture and Simulation of Chip Multi-Processors*, Nov 2005.

[21] 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 HPCA*, Feb. 2004.

[22] 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 HPCA*, Feb. 2005.

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

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

[25] S. Mutoh and et al. 1-V Power Supply High-Speed Digital Circuit Technology with Multithreshold-Voltage CMOSS. *IEEE Journal of Solid-State Circuits*, 30(8):847–854, Aug 1995.

[26] V. S. Pai. *Exploiting Instruction-Level Parallelism for Memory System Performance*. PhD thesis, Rice University, 2000.

[27] B. Sinharoy, R. Kalla, J. Tendler, R. Eickemeyer, and J. Joyner. Power5 System Microarchitecture. *IBM Journal of Research and Development*, 49(4), 2005.

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

[29] 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.

[30] 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.

[31] 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.

[32] G. E. Suh, S. Devadas, and L. Rudolph. A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning. In *Proceedings of the Eighth HPCA*, Feb. 2002.

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

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

[35] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum. Operating System Support for Improving Data Locality on CC-NUMA Compute Servers. In *Proc. of ASPLOS-7*, Oct. 1996.

[36] Virtutech AB. Simics Full System Simulator. http://www.simics.com/.

[37] Wisconsin Multifacet GEMS Simulator. http://www.cs.wisc.edu/gems/.

[38] 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.

[39] M. Xu, R. Bodik, and M. D. Hill. A Regulated Transitive Reduction (RTR) for Longer Memory Race Recording. In *Proc. of ASPLOS-12*, Oct. 2006.

[40] M. Zhang and K. Asanovic. Miss Tags for Fine-Grain CAM-Tag Cache Resizing. In *Proceedings of the International Symposium on Low Power Electronics and Design*, Aug. 2002.

[41] 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.

[42] 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.