MANAGING WIRE DELAY IN CHIP MULTIPROCESSOR CACHES

by

Bradford M. Beckmann

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN - MADISON

2006

Abstract

Increasing on-chip wire delay and growing off-chip miss latency, present two key challenges in designing large Level-2 (L2) CMP caches. Currently, some CMPs use a shared L2 cache to maximize cache capacity and minimize off-chip misses. Others use private L2 caches, replicating data to limit the delay from slow on-chip wires and minimize cache access time. Ideally, to improve performance for a wide variety of workloads, CMPs prefer both the capacity of a shared cache and the access latency of private caches.

In this thesis, we propose three techniques that combine the benefits of shared and private caches. In particular, to reduce access latency in a shared cache, we investigate cache block migration and on-chip transmission lines. Migration reduces access latency by moving frequently used blocks towards the lower-latency banks. We show migration successfully reduces latency to blocks requested by only one processor, but doesn't reduce the latency to shared blocks. In contrast, transmission lines can reduce on-chip wire delay by an order of magnitude versus conventional wires and provide low latency to all shared cache banks. We demonstrate on-chip transmission lines consistently improve performance versus a baseline shared cache, but bandwidth contention can limit them from reaching their full potential.

To improve the effective capacity of private caches, we propose Adaptive Selective Replication (ASR). ASR dynamically monitors workload behavior and replicates cache blocks only when it estimates the benefit of replication (lower L2 hit latency) exceeds the cost (more L2 misses). When ASR detects replication is less beneficial, processors coordinate writebacks with remote on-chip caches to conserve cache storage. ASR provides a robust CMP cache hierarchy: improving performance versus both shared and private caches. Additionally, ASR can leverage the fast remote cache access latency provided by transmission lines and reduce off-chip misses versus a design using conventional wires. We demonstrate the combination of transmission lines and ASR outperforms either isolated technique and preforms similarly to a shared cache using four times the transmission line bandwidth.

Acknowledgments

I dedicate this thesis to my wife, Jennifer. Her support over these six years has been invaluable. Though God has tremendously blessed me throughout my entire life, no gift has been greater than her. However, the space in this section is better spent acknowledging others, because words cannot describe how grateful I am of her.

I'm forever thankful to my advisor David Wood. I could not had asked for a better advisor. David is not only incredibly brilliant, but one of the most admirable people I have ever met. He taught me how to break down data and to write research papers that provided intuition and insight. I really appreciate all the hours he spent working with me and my research. Furthermore, I appreciate him making my graduate school experience very enjoyable. Now it is my job to remember what he has taught me.

I would like to thank the other members of my defense and preliminary exam committees, as well as other Wisconsin faculty members. First, I would like thank Mark Hill for his advice and helpful suggestions. His feedback certainly has improved both my research and presentation skills. Second, I would like to thank the other members of my defense and preliminary exam committees for their objective feedback: Guri Sohi, Mikko Lipasti, Mike Swift, and Charlie Chen. Also, I would like to thank some other Wisconsin faculty members that taught me: Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Ras Bodik, and Mary Vernon.

Similarly, I would like to thank all of the former and current members of Multifacet: Dan Sorin for initially encouraging me to pursue architecture research and allowing me to tag along with him to watch Danny Heatly and the 2000-2001 Badger hockey team; Milo Martin for showing me how to do just about everything research-wise including the proper way to program; Carl Mauer for answering what I am sure seemed to be an infinite number of questions; Alaa Alameldeen for listening to my concerns and thoughts, and providing me very useful feedback; Min Xu for always being a pleasant guy to talk to both profession-

ally and personally; Pacia Harper for acquainting me with Multifacet; Ross Dickson for introducing me to the Multifacet code base; Kevin Moore for being a good friend, giving me an outlet at work to discuss baseball and college basketball, and helping me professionally to communicate my ideas clearer; Jarrod Lewis for being a great intramural football quarterback; Bhavesh Mehta; Michael Marty for being a good friend, providing me some competition on the golf course, and for giving me helpful feedback that stimulated my research; Luke Yen for maintaining the Opal code base and helping me out whenever I asked; Jayaram Bobba; Michelle Moravan; Dan Gibson for being a fun guy to have a beer with and discuss ostriches; and Andy Phelps.

There are many other friends I thank as well. First I want to thank my good friend and roommate of four years, Tim Denehy. Despite the fact that Tim consistently dominated me on the golf course, he never gloated. Tim's talent both intellectually and athletically is only equaled by his modesty. Tim has been a great friend and a great asset professionally. Also I thank my other friends in Madison—the list is so long I fear to forget someone. Additionally, I thank the other things in Wisconsin that I'll miss: Lake Louie beer; Etes-Vous-Prets (EVP) coffee; Wisconsin Football, Women's Volleyball, Men's Basketball, and Men's Hockey teams; and the Big Ten Pub. Finally, I thank all my friends back in Ohio and woxy.com for broadcasting over the internet so I'd miss home a little less.

My family deserves a special thank you, especially my parents. Through my parents hard work, my life has been full of great opportunities and experiences. I hope one day to be as great of a role model to my kids, as they have been to me. Also I want to thank my sisters—Ellen, Jill, and Tricia—who have been tremendously supportive and my Uncle Mike who probably doesn't realize how much I enjoy spending time with him.

Finally, my research has been financially supported these different sources: the National Science Foundation with grants CCR-0324878, CNS-0205286, and EIA-9971256 and donations from IBM, Intel and Sun Microsystems.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past decade, computer-driven productivity and efficiency gains have lead to strong economic growth [51]. At the heart of this new computer infrastructure lie the servers that provide the backbone of electronic commerce and information distribution. In particular, these servers improve commercial workload efficiency by increasing throughput. Multiprocessor systems can increase server throughput because commercial workloads contain abundant parallelism.

With the increasing number of transistors available on a chip per process generation [82], multiprocessor systems have shifted from multi-chip systems to single-chip implementations. Specifically, chip multiprocessors (CMPs) containing 2-8 processors have recently became commercially available [53, 62, 65, 77]. In order to improve CMP performance, these CMPs require high-bandwidth low-latency communication between processors and their associated instructions and data.

By quickly providing processors with instructions and data, on-chip caches can significantly improve CMP performance. Small private high-level caches integrated closely with the processor cores provide each processor quick access to their most recently requested instructions and data. However, these finite-sized caches satisfy only a portion of requests, and many other requests must access larger lower-level caches. These large on-chip caches should both store a lot of data, thus minimizing off-chip miss latency's impact on performance, and quickly retrieve requested data to reduce global wire delay's effect on performance.

Low-level cache management presents a key challenge, especially in the face of the conflicting requirements of reducing off-chip misses and managing slow global on-chip wires. Current CMP systems, such as the IBM Power 5 [53] and Sun Niagara [62], employ shared caches to maximize the on-chip cache capac-

ity by storing only unique cache block copies. While shared caches usually minimize off-chip misses, they have high access latencies since many requests must cross global wires to reach distant cache banks. In contrast, private caches [65, 77] reduce average access latency by migrating and replicating blocks close to the requesting processor, but sacrifice effective on-chip capacity and incur more misses.

An attractive alternative to mitigating slow global wires with private caches, is improving shared cache access latency with technologies such as dynamic block migration and on-chip transmission lines. Cache block migration can improve shared cache performance by exploiting the fact that cache banks closer to a processor can be accessed more rapidly than distant banks. In particular, migration reduces average cache access time by moving frequently requested data to the closer cache banks. In contrast, on-chip transmission lines replace slow conventional wires with a high speed alternative. However, despite their substantial speed advantage—up to a factor of 10 by the end of the decade—transmission lines will not replace most conventional on-chip wires because they have over a factor of 10 lower bandwidth density. Instead, these sparse wires must be used carefully in order to improve shared cache performance.

In comparison, private cache performance can be improved using selective replication and transmission lines. In particular, selective replication strives to balance the latency benefit of replication with the capacity benefit of storing more unique block copies. By limiting replicas, selective replication increases the probability a hit is satisfied by a remote on-chip cache. Complementary, integrating transmission lines in the on-chip communication network can substantially reduce remote cache hit latency. Thus, together selective replication and transmission line can provide better private CMP performance than either technique in isolation.

Overall, this dissertation examines the performance impact of migration, selective replication, and transmission lines in the context of both shared and private CMP caches. Specifically, the dissertation focuses on CMPs with Level-2 (L2) cache banks comprising the lowest level of the cache hierarchy, but these techniques also apply to caches with greater depth. The dissertation not only evaluates important multithreaded

commercial workloads, but also selected multithreaded scientific workloads to demonstrate the different techniques' sensitivity to workload behavior. Due to the significant sharing that exists in commercial workloads, the dissertation illustrates that block migration is less effective for a shared CMP cache than previous results have shown for uniprocessors. Instead, the dissertation advocates for selective replication within a private CMP cache. By only allowing a few replications, selective replication can attain a majority of the benefit (reduce latency) without encountering most of the cost (increase off-chip misses). Furthermore, the dissertation proposes an adaptive selective replication mechanism because the optimal amount of replication varies depending on workload behavior and system constraints. Finally, the dissertation demonstrates on-chip transmission lines improve both shared and private cache performance and shows that transmission lines can work in concert with other techniques.

In the remainder of this chapter, Section 1.1 introduces on-chip global wire technology and Section 1.2 describes the multithreaded workload behavior that influences CMP cache design. Section 1.3 details the three wire delay management techniques analyzed in this dissertation. Section 1.4 presents the dissertation's main contributions, and Section 1.5 provides a roadmap for this document.

1.1 On-chip Global Wire Technology

On-chip wire delay plays an increasingly significant role in integrated circuit design. Design partitioning, the integration of more metal layers, and higher repeater density allow conventional wire dimensions to decrease slower than transistor dimensions. Thus, wire delay remains relatively constant for short intra-partition distances [46, 104]. However, on-chip wire delay between separate partitions is a growing performance bottleneck. For instance, global wire delay increased the L2 cache access latency for Intel's Prescott processor [91] from 23 cycles to 27 cycles when its L2 cache size increased from 1 MB to 2 MB [71, 93]. In the future, conventional global wire delay will become even worse. For example, transmitting data 1 cm

requires about 3 cycles in current (2006) technology, but will necessitate over 8 cycles in 2010 technology assuming a cycle time of 20 fanout-of-three delays [38].

An emerging alternative approach to conventional slow global wires is to use on-chip transmission lines [21]. Transmission lines exhibit much lower latencies than conventional wires since their signalling speed is dominated by a relatively short inductive-capacitance (LC) delay rather than a series of a relatively large resistive-capacitance (RC) delays. In layman's terms, one can compare the latencies of transmission lines and conventional RC wires by comparing the speed of a ripple moving across water in a bathtub to the speed of changing a bathtub's water level. While this gross analogy provides some insight behind transmission line's latency advantage, it ignores the key issues of realistic on-chip transmission lines, including attenuation.

In order to communicate the incident wave across on-chip global wires, signal attenuation must be minimized. In particular, transmission lines require very wide, thick wires and dielectric spacing to operate in the LC range, which are only available in a chip's uppermost interconnection metal layers. These extremely sparse metal layers are best utilized for the few long distance communication links whose latency can have a significant impact on overall system performance.

By targeting transmission lines to long cross-chip communication, they can facilitate compact layout, and reduce power consumption. Chapter 2 describes why transmission lines don't require repeaters like conventional wires. Also, using first-order equations, Chapter 2 demonstrates that transmission lines can consume less dynamic power than conventional interconnect. Later, Chapter 7 proposes using transmission lines to improve the layout and performance of several different CMP caches.

1.2 Multithreaded Workload Behavior

In order to improve CMP cache performance, it is important to understand the demand workloads place on them. In particular, this dissertation focuses on CMP cache designs that improve multithreaded commer-

cial and scientific workload throughput. The on-chip cache blocks of multithreaded workloads can be split into three different sharing types that exhibit distinct behavior: 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. The remainder of this section summarizes each sharing type's unique behavior. Later, Section 1.3 will describe how different techniques can exploit these behaviors to improve CMP cache performance.

Single Requestor Blocks. Single requestor blocks consume the majority of cache capacity for both commercial and scientific workloads. However, single requestor blocks only satisfy the majority of requests in scientific workloads. In general, data blocks and not instruction blocks account for almost all single requestor blocks stored in CMP caches. Because scientific algorithms successfully split their data sets into multiple independent segments, these workloads frequently utilize single requestor blocks. In contrast, commercial workloads often share L2 blocks between multiple threads, in part because these workloads have much larger instruction footprints [12]. Thus, techniques improving the latency to single requestor L2 blocks are more effective in improving scientific workload performance than commercial workload performance.

Shared Read-only Blocks. Shared read-only blocks consume relatively little capacity in a shared L2 cache, but satisfy a disproportionally large number of L2 requests. For commercial workloads, with instruction footprints that often spill into the L2 cache, the high utilization of shared read-only L2 blocks is especially evident. Interestingly, shared read-only requests exhibit high locality for these shared read-only L2 blocks. Specifically, for the evaluated commercial workloads, less than 300 KB of shared read-only data satisfies at least 70% of all shared read-only L2 requests. Some scientific workloads also frequently access shared read-only blocks, but these requests display significantly less locality and consume more L2 capacity than the commercial workloads. Therefore, techniques exploiting shared read-only request locality can improve commercial workload performance more than scientific workload performance.

Shared Read-write Blocks. For most commercial workloads, shared read-write blocks satisfy a noticeable proportion of requests, but for scientific workloads, they satisfy very few requests. Furthermore, for these commercial workloads, shared read-write blocks exhibit migratory sharing behavior [99] and average less than 2 intervening requests between writes. Therefore, a CMP cache that disallows shared read-write replication, could facilitate faster cache-to-cache transfers by removing coherence invalidations on writes. Overall, techniques that reduce shared read-write request latency would likely improve commercial workload performance more than scientific workload performance.

1.3 Wire Delay Management Techniques for Different Sharing Types

By utilizing different wire delay management techniques, CMP caches can exploit each sharing type's unique behavior. This section introduces three such techniques and provides their high-level intuition. These techniques will be described in greater detail through the dissertation's remaining chapters.

Migration for Single Requestor Blocks. By moving single requestor blocks closer to their requesting processor, migration can potentially reduce access latency without adversely affecting the other on-chip processors. Migration can be implemented statically by using a private CMP cache design, or can be preformed dynamically within a shared CMP cache. Dynamic migration was first proposed by Kim, Burger, and Keckler [58] within the context of a uniprocessor Non-Uniform Cache Architecture (NUCA). Specifically, Kim *et al.* exploited the fact that cache banks closer to the processor could be accessed faster than further banks and migrated frequently requested blocks to these closer banks. The resulting dynamic migration policy achieved substantial performance improvements for the uniprocessor.

In contrast, for a shared CMP cache, this dissertation demonstrates that migration is less effective because shared blocks tend to congregate in the middle of the cache. Furthermore, migration can create uneven cache bank utilization, thus, increasing cache conflict misses and negating migration's latency reduction. Using a high-level model, Chapter 4 examines directly migrating blocks to the last requesting processor's

closest cache bank. The model's results show migration could improve scientific workload performance because of frequent use of single requestor blocks, but migration potentially degrades commercial workload performance because of repeated requests to shared blocks. Then, using full system simulation, Chapter 5 demonstrates that a more gradual migration policy is also ineffective. Instead, Chapter 6 shows a private CMP cache that statically allocates single requestor blocks to the requesting processor's local cache banks is more effective. However, replication between private caches can reduce effective cache capacity and negate migration's latency benefit.

Selective Replication for Shared Read-only Blocks. By intelligently replicating data between private L2 caches, selective replication can reduce shared read-only request latency, without adversely affecting cache capacity. By migrating and replicating data, private caches minimize cache access time. However, excessive replication sacrifices on-chip capacity and incurs more off-chip misses. Recently, researchers have proposed three different selective replication schemes that strive to balance latency and capacity [20, 26, 121]. Two of these previous proposals—CMP-NuRapid [26] and Victim Replication [121] use static rules that can not adjust to changes in workload behavior, while the third—Cooperative Caching [20]—controls replication capacity via a parameterized value. Section 6.5.1 of Chapter 6 describes these proposals in detail. Similar to Cooperative Caching, Chapter 6 proposes the Selective Probabilistic Replication (SPR) mechanism that also controls replication via a parameterized value, but does so with less hardware. Also, Chapter 6 proposes the Adaptive Selective Replication (ASR) mechanism that dynamically monitors workload behavior to control SPR's replication parameter. ASR improves performance versus the recent hybrid proposals and provides performance stability by always performing at least comparably to the best alternative.

Transmission Lines for Shared Read-write Blocks. Transmission lines' significant speed advantage over conventional wires can reduce shared read-write request latency, as well as overall request latency. For a shared CMP cache, Chapter 7 demonstrates transmission lines can reduce overall L2 hit

latency versus conventional wires, but bandwidth contention limits their overall effectiveness. Also Chapter 7 shows transmission lines can improve private CMP cache performance. For the private cache, transmission lines are most effective when they target low-bandwidth latency-critical signals such as request messages.

1.4 Thesis Contributions

The main contributions of the dissertation are:

- **Multithreaded Workload Characterization.** The dissertation breaks the working sets of commercial and scientific workloads into three different sharing types: single requestor data, shared read-only data, and shared read-write data. The characterization shows that each sharing type displays distinct behavior that can be exploited using different caching techniques.
- **Block Migration in CMP Caches.** The dissertation illustrates that block migration is less effective for CMPs than previous results have shown for uniprocessors. Even with a perfect search mechanism, migration alone only improves performance versus a shared cache baseline for four workloads—Jbb, Art, Barnes, and Ocean—by a maximum of 2%, while degrading the performance of the other four workloads by as much as 7%. This is in part because shared blocks migrate to the middle equally-distant cache banks, accounting for 55-83% of L2 hits for the commercial workloads.
- **Selective Probabilistic Replication (SPR).** The dissertation presents *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 previous proposals CMP-NuRapid [26] and Cooperative Caching [20], and equivalent hardware to the Victim Replication proposal [121].

- **Adaptive Selective Replication (ASR).** The dissertation proposes the *Adaptive Selective Replication (ASR)* mechanism that dynamically controls replication within a private CMP cache. 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.
- **Transmission Line Caches.** The dissertation demonstrates transmission lines reduce on-chip cache access latency for both a shared and private CMP cache by as much as 30%. Overall, these Transmission Line Caches (TLC) improve performance up to 8% using single bit transmission lines and 15% for transmission lines using wave division multiplexing.
- **Combination of Techniques.** The dissertation shows the combination of migration, ASR, and transmission lines achieves 14% average performance improvement over an 8-banked baseline shared cache and performs competitively with a 64-banked TLC using four times the transmission line bandwidth.

Prior versions of the block migration and transmission line cache work described here have been previously published in conjunction with my advisor David Wood [14, 15]. The work of Selective Probabilistic Replication and Adaptive Selective Replication was developed jointly with Michael Marty and David Wood and will be published soon [13].

1.5 Dissertation Structure

The thesis begins with Chapter 2 summarizing future on-chip wire technologies with a focus towards on-chip transmission lines. Next, using an analytical model, Chapter 3 analyzes how wires impact CMP cache design and presents equations describing memory system performance for both shared and private cache hierarchies. Chapter 4 characterizes multithreaded workload behavior and then uses the characterization data to explore the potential performance benefits of migration and selective replication. Following the

high-level exploration, Chapter 5 use full system simulation and performs detailed analysis of cache block migration within a shared CMP cache. Then Chapter 6 proposes the Adaptive Selective Replication mechanism that dynamically regulates replication within a private CMP cache to match workload demand. Chapter 7 presents a multitude of cache organizations that utilize on-chip transmission lines to improve performance, including a private cache design utilizing ASR. Finally, Chapter 8 concludes the dissertation and provides several avenues for future work.

Chapter 2

Background: On-Chip Wire Technology

This chapter presents an overview of on-chip wire technology. Section 2.1 and Section 2.2 describe the delay, physical requirements, and power characteristics of conventional on-chip communication and on-chip transmission line technology, respectively. Then, Section 2.3 quantitatively compares these three characteristics for both wire technologies. Finally, Section 2.4 discusses how other potential wire technologies—such as package level interconnects and on-chip optical communication—appear similar to on-chip transmission lines from an architectural perspective.

2.1 Conventional RC Communication

The vast majority of on-chip communication is best described using conventional resistive-capacitance (RC) models. This section describes the propagation delay, physical requirements, and power consumption of these wires.

2.1.1 Propagation Delay

Resistance, capacitance, and inductance determine wire delay. Equation 2.1 presents the telegrapher's equation that describes voltage propagation across any type of wire as a function of time assuming conductance is negligible:

$$\frac{\partial^2 V}{\partial x^2} = RC\frac{\partial V}{\partial t} + LC\frac{\partial^2 V}{\partial t^2} \tag{2.1}$$

where x is the distance along the wire, t is time, V is voltage, and R, C, and L represent the resistance, capacitance, and inductance of the wire. Wire resistance is directly proportional wire length and is inversely proportional to a wire's cross-sectional area. Meanwhile, wire capacitance is directly proportional to a wire's surface area and intermetal dielectric constant, and is inversely proportional to the distance between wires. Finally, wire inductance depends on the rate of current change and circuit layout.

For conventional on-chip wires, with relatively small cross-sectional area, the first RC term dominates. In other words, voltage *diffusion* across the wire determines signaling speed [28]. Furthermore, as wire distance increases, both wire resistance and capacitance increases because both terms are directly proportional to wire length. The result is conventional wire delay grows quadratically with distance.

To control wire delay across long on-chip distances, designers insert repeaters, e.g. inverters, to break long wires into multiple shorter segments. For these segmented links, wire delay grows linearly rather than quadratically with distance [116]. However, increasing wire density and operational frequencies dictate increasing number of repeaters, leading to three key problems [46]:

- Repeaters require a substantial amount of area for their large transistors.
- Repeaters necessitate disciplined floorplanning to allocate the necessary substrate area at the proper locations.
- Repeaters need many via cuts from the upper metal layers down to the substrate, which congest the interconnection layers below and reduce the overall wire bandwidth.

Due to repeater insertion, determining the RC delay for conventional global communication not only requires determining the resistance and capacitance of their actual wire segments, but also must include the parasitic terms of their intermediate drivers and receivers. Amrutur and Horowitz [4] present the following a simple approximation equation (Equation 2.2) that incorporates these parasitic capacitances to determine the total delay of a signal traveling across a single wire segment:

TABLE 2-1. ITRS Projections for Conventional Global Wires

Year	**2006**	**2007**	**2008**	**2009**	**2010**
Technology (nm)	**78**	**68**	**59**	**52**	**45**
Reachable Distance per cycle (mm)	2.3	1.6	0.9	0.6	0.4
Delay / 10 mm (ns)	1.65	2.09	3.16	4.10	5.23
Minimum wire pitch (nm)	250	210	177	156	135
Intermetal dielectric constant (κ)	2.7	2.4	2.4	2.2	2.2

$$\text{Conventional RC Communication Delay} = R_d(C_r + C_w) + p + R_w\left(\frac{C_w}{2} + C_r\right) \quad (2.2)$$

where R_d and R_w are the driver and wire resistances, C_w and C_r are the wire and receiver capacitances, and p is the intrinsic delay of the driver due to its junction capacitance.

The performance of global wires using repeaters doesn't scale compared to transistors. Specifically, as transistors become smaller and faster, wire pitch is forced to scale to smaller geometries. However, as previously mentioned, reducing a wire's cross-sectional area, increases the wire's RC delay. To combat this problem, manufacturers integrate an additional metal layer per process technology to reduce wire density demand and utilize lower-k dielectrics to reduce wire capacitance [39]. These manufacturing enhancements have allowed global wire delay to stay relatively constant in terms of reachable transistors per cycle across technology generations [104]. However, Table 2-1 shows the International Technology Roadmap for Semiconductors (ITRS) [39] projects the reachable absolute distance per cycle will increase by over 5 times by 2010 for global RC wires. To make communication delay scale with transistors, designers must look beyond conventional wires.

2.1.2 Physical Requirements

Conventional global wire pitch scales relatively well, but their intermediate repeaters and latches necessitate significant substrate area. Table 2-1 shows the minimum wire pitch of conventional global wires scales well with transistor dimensions, with both shrinking by approximately 45% over the next five years. However, as improving technology integrates more processor cores on chip, global interconnect bandwidth must increase. Furthermore, as frequency increases and wire dimensions decrease, global wires require higher repeater density. Frequently repeated global interconnect requires substantial dedicated substrate area and thus cannot be routed over other large structure. For instance, Kumar, Zyuban, and Tullsen [66] demonstrated for an eight processor CMP implemented in 65 nm technology, global interconnect using conventional wires consumes the equivalent die area of three IBM Power4-like [106] cores. Overall, the substrate area dominates the physical requirements of conventional global interconnect.

2.1.3 Power Consumption

Due to large wire capacitance, dynamic power dominates the total power consumed by conventional interconnect. Conventional RC interconnect consumes dynamic power by charging and discharging the capacitance of each wire segment from one voltage value to another. Specifically, Equation 2.3 shows conventional RC communication dynamic power equals the power required to change the voltage, *V*, across the wire's total capacitance, *C*, for a given frequency, *f*, and data activity factor, α [103]:

$$\text{Conventional RC Communication Dynamic Power} = \alpha \times C \times V^2 \times f \tag{2.3}$$

Technology and microarchitecture innovations affect each term of Equation 2.3 to different degrees. For example, Intel's Foxton technology [86] dynamically adjusts the voltage and frequency by 50% and 80% respectively to ensure the Montecito processor stays within its power and thermal envelope. Ramprassad,

Shanbhag, and Hajj [90] present a source-coding framework that reduces the average activity factor by 36% for address and data buses, resulting in a 36% reduction in dynamic power. In contrast, ITRS [39] (Table 2-1) projects the intermetal dielectric constant (and wire capacitance $C = \kappa d / Area$) will only decrease by at most 23% over the next 5 years. Meanwhile, capacitance dependence on distance (d) cannot be reduced. Consequently, wire capacitance remains the hardest dynamic power term (Equation 2.3) to optimize for conventional global interconnect.

2.2 On-chip Transmission Line Communication

Transmission lines are an alternative wire technology with potentially lower communication delay. Printed-circuit board and other off-chip wire technologies are commonly designed to behave as transmission lines [28]. Conversely, although on-chip transmission lines have been explored for over 20 years [107], most on-chip wires using CMOS technology are designed to operate as lossy RC lines [112]. However, with improving fabrication technology, on-chip transmission lines are starting to emerge in CMOS circuits. For example, several current high performance chips use 7.5 mm transmission lines for clock distribution [80, 110, 117]. Longer (> 10 mm) transmission lines have been shown to work on CMOS test chips using very wide wires [21, 52] or low operating temperatures [31]. With the introduction of lower-k dielectrics [17] and increasing on-chip frequencies [47], more practical on-chip transmission lines will be available before the end of the decade. This section describes the propagation delay, physical requirements, and power demands of on-chip transmission lines.

2.2.1 Propagation Delay

Ideally, the inductance-capacitance (LC) product of transmission line wires determine their delay. Going back to the telegrapher's equation (Equation 2.1), this means that the second, LC term dominates wires with low resistance and high operating frequency. In other words, wave propagation determines signaling

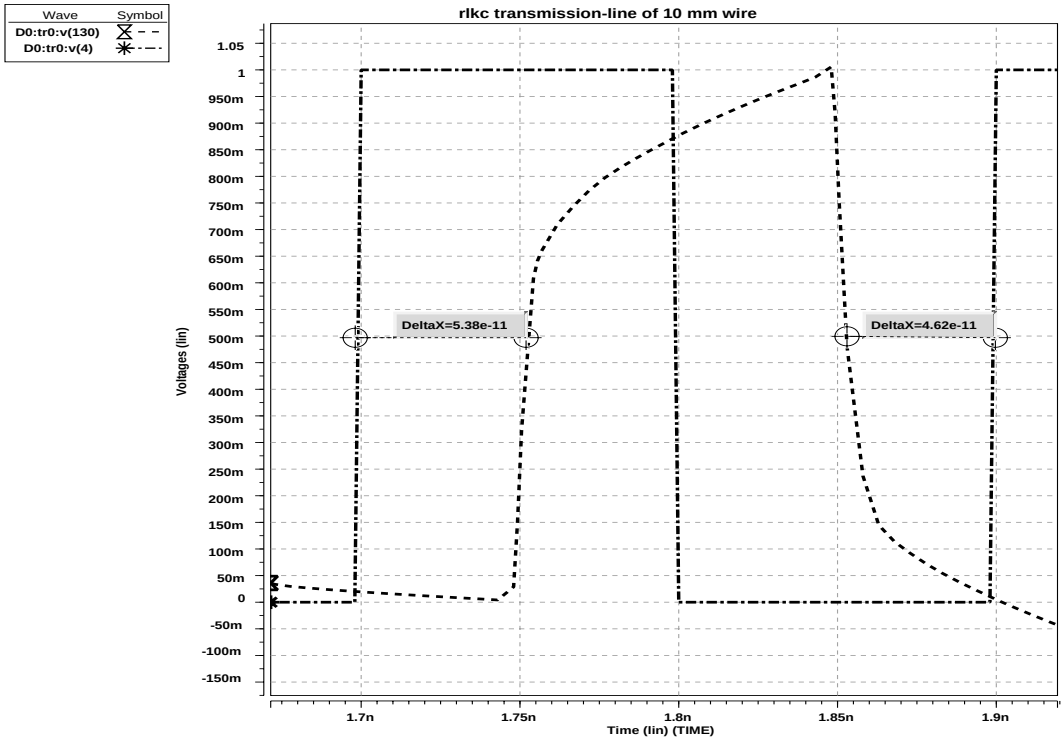


FIGURE 2-1. Sample Output Waveform of a 10 mm On-chip Transmission Line

speed [28]. Specifically, if signal frequency is greater than the cutoff frequency ($0.2R/L$), only the speed of light in the intermetal dielectric ($c_0/\sqrt{\varepsilon_r}$) limits signal speed, where R equals the transmission line resistance, and L depicts the transmission line inductance, c_0 represents the speed of light in a vacuum, ε_r corresponds to the intermetal dielectric's relative dielectric constant [49].

The resistance across an on-chip transmission line attenuates the incident wave traveling from the driver to receiver. In addition, the resistance is frequency-dependent due to the "skin effect". The skin effect phenomenon arises because at high frequencies, magnetic repulsion forces current towards the perimeter of the conductor, thereby reducing the wire's effective cross section. The higher frequency sinusoids composing a digital pulse cause the received signal to appear rounded and stretched out.

We utilized a two-dimensional field-solver program and circuit simulation to determine the delay of on-chip transmission lines because simple equations cannot model the skin effect accurately. We started by using Linpar [33], a two-dimensional field-solver program, to extract the inductance, resistance and capac-

itance characteristics of on-chip transmission lines. Using the resulting RLC matrices, we used HSPICE to simulate 5 GHz pulses travelling across the transmission lines [7]. Specifically, we modeled the transmission line's frequency dependent attenuation with HSPICE's W element transmission line model. We simulated four signal wires with shielding wires separating each of them under worst case signalling conditions. For example, Figure 2-1 illustrates that a 10 mm on-chip transmission line achieves about a 50 ps latency between the driver and receiver. The driver and receiver add an additional 110 ps of delay [21].

2.2.2 Physical Requirements

The on-chip transmission lines require wide metal pitch and dielectric spacing in order to achieve good incident wave propagation qualities. Specifically, due to their long length, transmission lines require thicker and wider metal tracks to maintain low wire resistance. Additionally, transmission lines necessitate thicker intermetal dielectrics to reduce their capacitance. While transmission line dimensions exceed the dimensions proposed for future conventional interconnect, they actually compare to the upper metal layers of previous high performance processors [16] and current silicon microwave chips [88].

Because transmission lines can quickly communicate across long distances without using repeaters, they facilitate more efficient layout. As previously mentioned, conventional RC communication requires vias and substrate area to access repeaters that amplify a signal as it travels across multiple wire segments. On the other hand, transmission lines do not require repeaters, and their lack of intermetal layer congestion and intermediate substrate area partially compensates for their greater routing dimensions.

We propose using single-ended voltage-mode signaling to communicate across on-chip transmission lines. Single-ended voltage-mode signalling propagates voltage pulses across a single point-to-point link. To reduce reflection noise across these relatively low loss transmission lines, we assumed source-terminated drivers with digitally-tuned resistance [28]. Receivers use a large input impedance termination for full wave reflection of the received signal. This design allows for the signal to significantly attenuate while

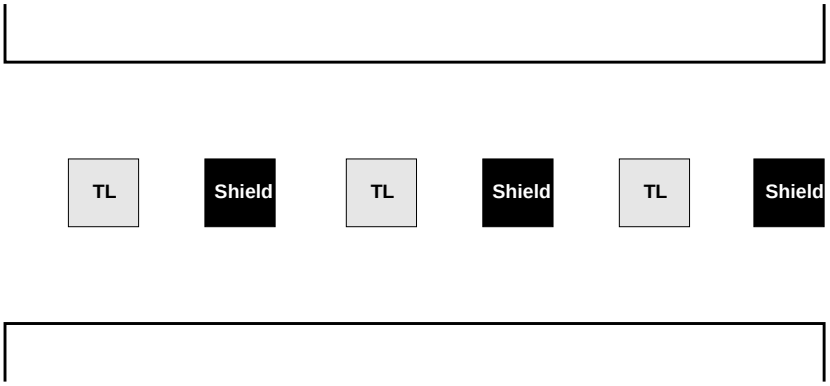


FIGURE 2-2. Stripline Transmission Lines

travelling across the wire, while the receiver can still receive a full signal swing (Figure 2-1). However this design does sacrifice noise immunity.

Single-ended voltage-mode signaling's increased susceptibility to noise requires additional routing area. To reduce the noise susceptibility, we propose using alternating power and ground shielding [57] lines between each transmission line (Figure 2-2). Also these transmission lines must be laid out in stripline fashion with a reference plane both above and below the transmission line metal layer to provide low resistance return paths for inductive induced currents [87]. Laying out the lines in this manner not only provides several individual low-resistive return paths, but also isolates each line from most capacitive and inductive cross-coupling noise. Additional enhancements such as low-swing differential signalling [119] and current-mode signalling [73] further improve noise immunity, but cost routing area, circuit complexity, and static power.

2.2.3 Power Consumption

Similar to conventional RC interconnect, dynamic power dominates the total power consumed by single-ended voltage-mode signaling. In voltage-mode transmission line signalling, the dynamic power equals the power required to create the incident wave (to the first-order). At the driver, the transmission line appears as a resistor equal to the characteristic impedance of the line. Therefore, the power supplied by the driver is

determined by voltage across its internal resistance, R_D, in series with the transmission line's characteristic impedance, Z_0, for the duration of the signal pulse, t_b [28]:

$$\text{Single-ended Voltage-mode Transmission Line Dynamic Power} = \alpha \times t_b \times \frac{V^2}{(R_D + Z_0)} \times f \tag{2.4}$$

For source-terminated transmission lines, $R_D = Z_0$, and for 50% duty cycle communication, $t_b = 1/(2 \times f)$. Thus, to the first-order, single-ended voltage-mode transmission line dynamic power depends only on voltage, characteristic impedance, and the activity factor, and is independent of frequency. Furthermore, these transmission lines only consume substantial power when actively transmitting data. In contrast, low-swing differential signalling [119] and current-mode signalling [73] always consume power for reference or bias voltages. Powering down these transceivers during periods of inactivity could reduce their constant power consumption [96]. However, these complicated techniques sacrifice performance. In this thesis, we assume single-ended voltage-mode transmission lines because they best match low-utilized on-chip global signals.

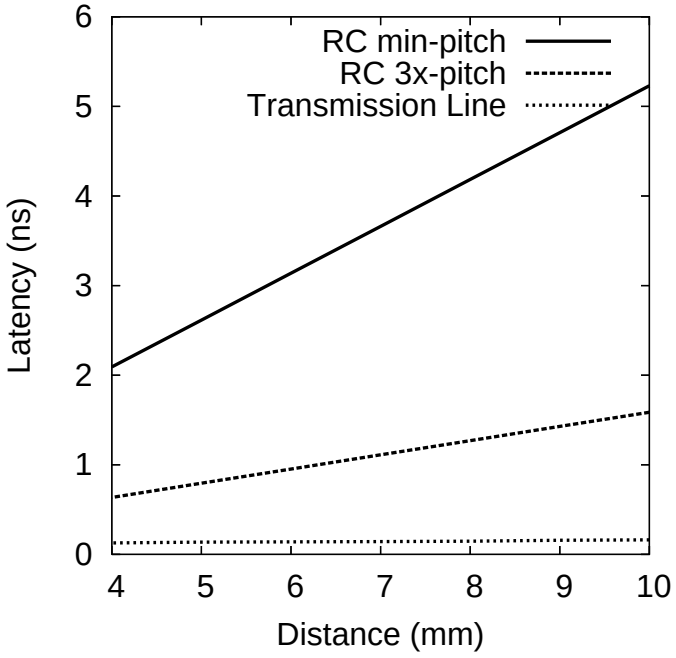


FIGURE 2-3. Latency Comparison

2.3 Comparison: Conventional RC Wires versus On-chip Transmission Lines

2.3.1 Latency

On-chip transmission lines offer a 6-19 times latency improvement versus conventional RC communication for global distances—4-10 mm—in 45 nm technology. Figure 2-3 plots the latency of minimum-pitched global RC interconnect, RC interconnect with 3x minimum-pitched dimensions[1], and on-chip transmission lines across global distances. Transmission lines achieve such a tremendous performance improvement not only due to their faster signaling speed, but also their lack of intermediate repeaters removes significant overhead compared to conventional RC interconnect.

1. Three times the minimum-pitch is often the largest permitted scale-up factor [104].

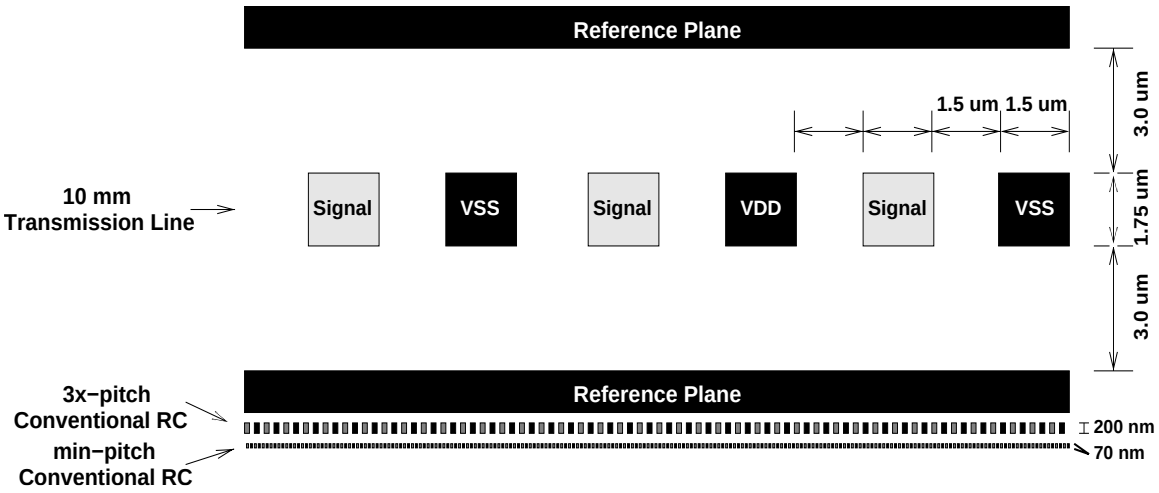


FIGURE 2-4. Cross-sectional Wire Comparison

2.3.2 Bandwidth Density

On-chip transmission lines demand significantly wider wires and intermetal spacing as compared to conventional communication. Figure 2-4 illustrates transmission lines require a substantial increase in wire area versus conventional interconnect and Table 2-1 compares transmission line bandwidth per width versus conventional RC communication. Specifically, on-chip transmission lines sacrifice 36-57 times the bandwidth density versus minimum-pitch global RC communication and 12-38 times the bandwidth density versus 3x-pitch global communication. The degree of bandwidth density lost depends on distance because longer transmission lines require larger dimensions in-order to operate in the LC range. Overall, these projected transmission line dimensions compare to current test chips dimensions [21, 52].

TABLE 2-1. Bandwidth Density Comparison

Wire Type	Width	Spacing	Height	Thickness	Normalized Bandwidth Density
min-pitch RC	35 nm	35 nm	70 nm	70 nm	1
3x-pitch RC	105 nm	105 nm	200 nm	200 nm	0.33
TL (9 mm)	1.25 μm	1.25 μm	1.75 μm	3.0 μm	0.056
TL (10 mm)	1.50 μm	1.50 μm	1.75 μm	3.0 μm	0.047
TL (11 mm)	1.75 μm	1.75 μm	1.75 μm	3.0 μm	0.040
TL (13 mm)	2.00 μm	2.00 μm	1.75 μm	3.0 μm	0.035

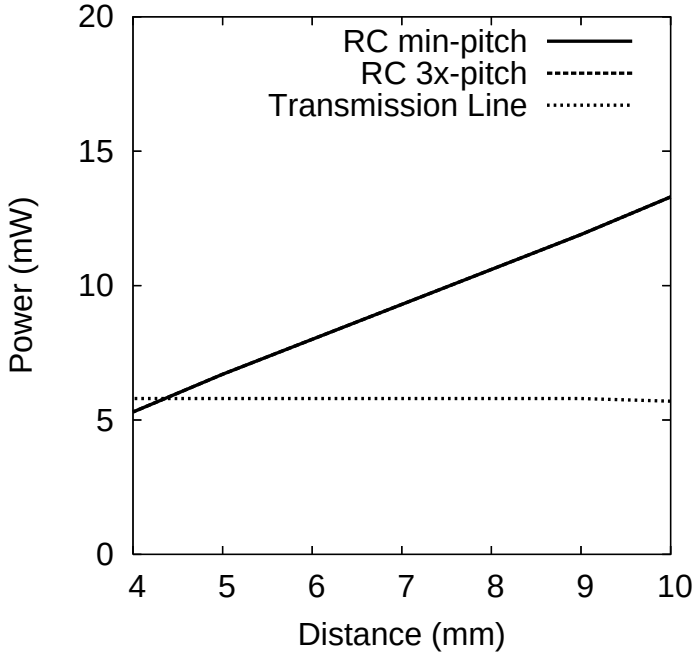


FIGURE 2-5. Dynamic Power Comparison

2.3.3 Dynamic Power

For long global on-chip communication, source-terminated voltage-mode transmission lines consume less dynamic power than conventional RC interconnect. By comparing Equation 2.3 to Equation 2.4, one sees that when $t_b / (2 \times Z_0) < C$, source-terminated transmission lines will consume less dynamic power than conventional interconnect. Figure 2-5 graphically shows for a 5 GHz clock frequency, this relationship

holds for global links beyond ~5 mm in length. (Note the RC min-pitch and RC 3x-pitch lines overlap because both have nearly the same capacitance and use the same repeater sizing rules [10])

2.4 Attractive Alternative Technologies

On-chip transmission lines are not the only technology to provide ultra-fast cross-chip communication, package level [9, 30, 29] and optical [23, 54, 60, 79] interconnects offer attractive alternative fast media. Package-level interconnect, like on-chip transmission lines, utilize transmission line signalling, but instead of implementing the wires within the on-chip metal layers, they utilize wires in a die-sized substrate above the processor die. The major advantage of package-level transmission lines is manufacturing cost. Both Wafer Level Package (WLP) [9] and Multi-chip Module (MCM) [29, 30] technology provide cross-chip transmission line propagation without adding costly die manufacturing steps. However, both technologies, sacrifice at least twice the bandwidth density as compared to on-chip transmission lines.

Similarly, optical communication [23, 54, 60, 79] offers near speed-of-light communication latency, but their waveguide requirements and power consumption limit their applicability. On-chip optical communication relies on approximately 1 μm wavelength light propagating through an approximate 4 μm-pitched on-chip waveguide. While these waveguide sizes are twice that of transmission lines, Wave Division Multiplexing (WDM) may provide higher bandwidth density than even conventional RC interconnect for communication links greater than 5 mm [23, 60]. Another current limitation of optical communication is power. Current optical transmitters consume more than 20 times the power than conventional RC interconnect [23]. However, future technology enhancements may substantially reduce optical communication's power consumption. Specifically, Chen *et al.* [23] project optical communication will consume less energy than conventional RC communication for 10 mm links in 45 nm technology.

2.5 Summary

While only time will tell what wire technology will be integrated into future CMPs, the deficiency of conventional wires communicating long distances is clear. All potential replacement technologies discussed in this section have the same general characteristics. All offer communication latency near the speed-of-light, none require intermediate vias or repeaters, and all offer potential power savings versus conventional RC communication for long distances. However, all suffer from different degrees of bandwidth density reduction, greater integration complexity, and higher manufacturing cost. Chapter 7 focuses on the architectural implications of these technologies with a specific focus on single-ended voltage-mode transmission lines because they have attained the most success in test chips. However, many of the Chapter 7's contributions apply to all mentioned alternative communication technologies.

Chapter 3

Global Wires and Large On-chip Caches

This chapter investigates wire technology's impact on large on-chip caches. First, Section 3.1 discusses how the growing disproportionate relationship between on-chip wire performance and transistor performance increases cache partitioning. Then, Section 3.2 uses a simple analytic model, called the Cache Investigative Model (CIM) to demonstrate how partitioning and wire technology affects cache latency and bandwidth. Finally, Section 3.3 introduces equations that relate the two baseline CMP cache organizations—shared and private—with overall memory system performance.

3.1 Wire Delay and Cache Partitioning

Both localized (< 1 mm) and global (> 1 mm) wire delays impact the design of large on-chip caches. Currently level-2/level-3 on-chip caches are divided into multiple banks and sub-arrays to optimize the individual bank's area/delay tradeoff [64, 78]. In the future, large on-chip caches will be partitioned into smaller banks so that local wires [113] match increasing SRAM density [8].

As cache organizations move towards smaller banks, global wire delay between cache banks becomes a more dominant performance bottleneck. Currently, designers split large caches into 3 [53] to 4 [63] independently addressable cache banks and use a crossbar to provide the on-chip processors uniform cache access time. However, crossbar latency and bandwidth will not scale to future generation CMPs [66]. Kim *et al.* [58] addressed this problem by defining a family of Non-Uniform Cache Architecture (NUCA) designs. Similar to Figure 3-1, all practical NUCA designs assume a 2D array of independently address-

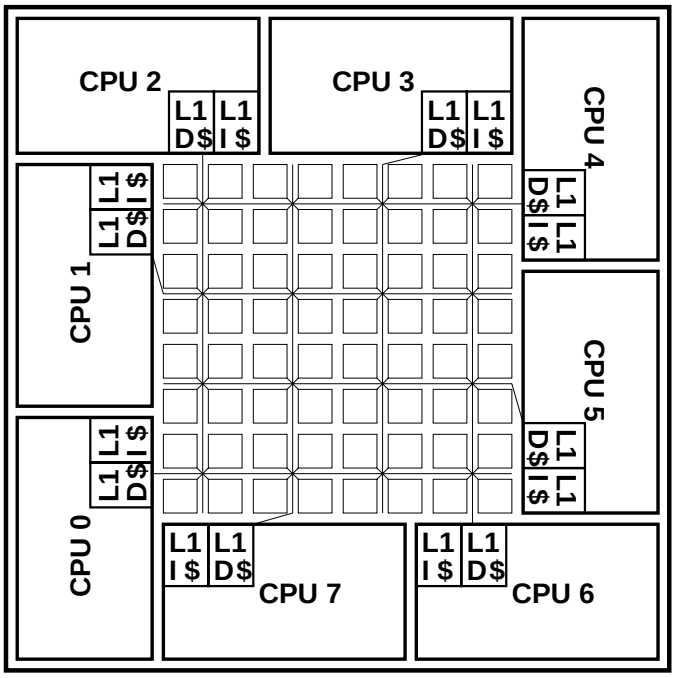


FIGURE 3-1. NUCA Cache Network for an 8 Processor CMP

able cache banks accessed via a switch interconnect. Thus, cache access latency is non-uniform with each on-chip processor observing varying latency depending on the distance to the cache bank it's accessing.

The optimal NUCA network configuration depends on the relationship between cycle time and global wire latency and bandwidth. For example, conventional RC wires provide relatively high latency and bandwidth. Assuming an aggressive 8 FO4 cycle time [47] for the 45 nm technology, Kim *et al.* [58] determined a 256 banked NUCA cache provided the optimal balance between intra-bank and inter-bank latency. However, due to recent studies advocating that power constraints will limit future frequency scaling [102], We assume a slower 20 FO3 [38] cycle time. The slower frequency moves the optimal NUCA configuration using conventional RC interconnect to a design with fewer banks. Additionally, on-chip transmission lines further move the optimal NUCA configuration to even fewer partitions because their lower bandwidth density limits network connectivity.

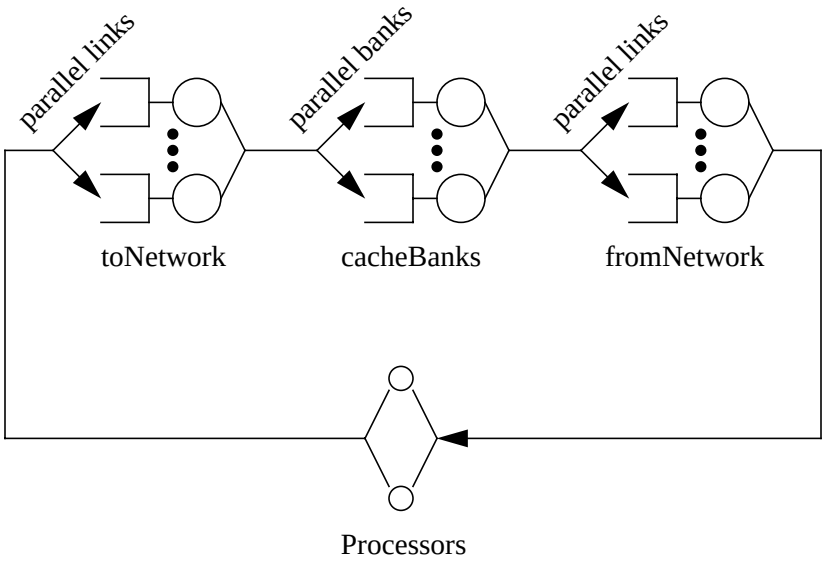


FIGURE 3-2. Diagram of the Cache Investigation Model

The next section illustrates this relationship between the optimal NUCA network configuration and global wire latency and bandwidth using a high-level analytic model.

3.2 CIM: Cache Investigative Model

This section provides a closed Approximate Mean Value Analysis (AMVA) model to illustrate the global wire's impact on the optimal cache design. The model, called the Cache Investigative Model (CIM), investigates how cache bank partitioning in conjunction with global wire latency and bandwidth affects the average access time of a large on-chip cache. CIM does not strive to accurately predict the actual cache performance, but instead CIM captures the high-level cache design tradeoffs.

CIM uses three queuing centers and one delay center to model a simplified cache network (Figure 3-2). The three queuing centers represent the network from the processors to the cache banks, the network from the cache banks back to the processors, and the actual cache banks. All network links and switches are assumed to be fully pipelined. Therefore, the service time for the network queuing centers equals the aver-

age number of link-switch pairs traversed times two. The service time for the cache bank queuing centers equals the bank access time. CIM assumes a uniform demand for the networks and cache banks. The delay center approximates the time between processor cache requests.

The model's customers represent round-trip cache messages. To account for writeback bandwidth, the number of customers equals the number of processors multiplied by the maximum requests per processor times 2. Since the bandwidth required by a request and subsequent response is similar to the bandwidth required by a writeback and subsequent acknowledgement, CIM treats them equivalently. CIM's closed model captures the dependence between cache access time and throughput.

CIM's overall goal is determining the average cache access latency of a large multi-banked cache. Equation 3.1 shows that the average cache access latency equals the summation of the calculated residence times at the three service centers:

$$\text{Average Cache Access Latency} = R_{toNetwork} + R_{cacheBanks} + R_{fromNetwork} \tag{3.1}$$

Equation 3.2 determines the residence time at each service center:

$$R_k = \begin{cases} D_{processor} & \text{(processor delay center)} \\ D_k(1 + A_k) & \text{(queuing centers)} \end{cases} \tag{3.2}$$

Where D_k equals the inputted service time of the delay or queuing center and A_k equals the calculated average number of customers seen at center k when a new customer arrives [68]. Specifically, $D_{processor}$ equals the average think time at the processor delay center, $D_{toNetwork}$ and $D_{fromNetwork}$ equal the average number of link and switch pairs crossed (*avg. # of link-switch pairs*) to/from the processor from/to the cache bank, and $D_{cacheBank}$ equals the average bank access time.

Equation 3.3 estimates A_k using the calculated average number of customers at each queuing center, Q_k, and N equals the inputted total number of customers:

$$A_k = \frac{N-1}{N} Q_k \tag{3.3}$$

Equation 3.4 applies Little's Law to the whole queuing network to compute total system throughput, X:

$$X = \frac{N}{\sum_{k}^{K} R_k} \tag{3.4}$$

Where k ranges through all K service centers including the processor delay center.

Equation 3.5 applies Little's law to each queuing center individually to calculate Q_k:

$$Q_k = X R_k \tag{3.5}$$

Then, the AMVA model iterates between Equation 3.2 through Equation 3.5 until successive calculations of Q_k agree within a tolerance of 0.1%.

Finally, Equation 3.6 presents the average time between processor requests:

$$\text{Average Time Between Processor Requests} = \frac{D_{processor}}{N - \sum_{k}^{J} Q_k} \tag{3.6}$$

, where k ranges only through the J queuing centers,

TABLE 3-1. CIM: Cache Bank Partitioning Parameters

Parameter	8 Banks	16 Banks	32 Banks	64 Banks	128 Banks	256 Banks
Parallel Links	8	8	8	8	8	8
Avg. # of link-switch pairs	3	3	4	4	5	5
D_{bank} *(cycles)*	20	15	12	9	6	4
$D_{processor}$ *(cycles)*	100	100	100	100	100	100
N	2-256	2-256	2-256	2-256	2-256	2-256
Saturation pt. (cycles)	2.5	0.94	1.0	1.0	1.25	1.25

Equation 3.7 provides the saturation point where D_{max} equals the maximum of $D_{processor}$, $D_{toNetwork}$, and $D_{fromNetwork}$, using the inputted number of parallel servers.

$$\text{Satruation point} = \frac{D_{max}}{\text{\# of parallel servers}} \tag{3.7}$$

3.2.1 CIM: Cache Partitioning

This subsection demonstrates how increasing CMP cache partitioning affects cache access latency under high bandwidth demands. Specifically, for a level-2 CMP cache servicing eight on-chip processors, average inter-arrival times for L2 requests can be as low as 2 cycles for bursts as long as 1000 cycles. Therefore, the number of customers (*N*) vary between 2 and 256 in-order to analyze average cache access latency (Equation 3.6) under periods of high demand. The non-shaded rows of Table 3-1 present CIM's inputted parameters for an eight-processor CMP cache partitioned into 8 to 256 cache banks. The network between processors and banks represents the average number of links-switch pairs traversed where links are wide enough to transmit any message in a single cycle. Increasing bank partitioning reduces the size of each individual bank, and thus bank service time (D_{bank}) deceases from 20 cycles to 4 cycles [1]. However,

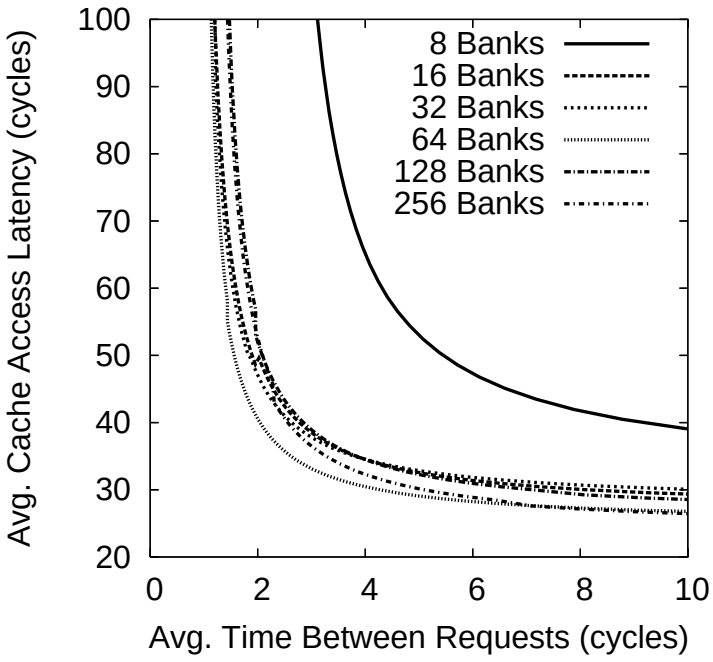


FIGURE 3-3. CIM: Cache Partitioning - Cache Access Time vs. Bandwidth Demand

more banks increase the *average number of link-switch pairs* traversed between processors and banks from 3 to 5.

CIM's investigation of cache partitioning exhibits the balance between the number of cache banks and average cache access time. For each cache design, Figure 3-3 plots the average cache access time versus bandwidth demand. Under high bandwidth demand—average time between requests less than 8 cycles—the 8 bank configuration encounters a dramatic increase in average cache access latency due to bank contention. In contrast, the other bank configurations encounter less than 15% increase in their average cache access latency until the average time between requests reaches 4 cycles. Between 2-4 cycles, the 64 bank configuration achieves the lowest latency. Also the bottom shaded row of Table 3-1 presents the saturation point for each cache design. All cache designs except for the 8-banked design achieve a saturation point of 1.25 cycles or less.

TABLE 3-2. CIM: Wire Technology Parameters

Parameter	Wide-Slow	Thin-Fast
Parallel Links	8	5
Avg. # of link-switch pairs	4	1
Link + switch latency (cycles)	1 + 1	5 + 1
D_{bank} *(cycles)*	9	9
$D_{processor}$ *(cycles)*	100	100
N	2-256	2-256
Saturation pt. (cycles)	1.0	2.0

3.2.2 CIM: Wire Technology

This sub-section demonstrates on-chip communication innovations will improve average cache access latency unless bandwidth limitations become a bottleneck. Specifically, the sub-section compares a 64-banked cache using wide and slow interconnect (wide-slow) to a 64-banked cache using thin and fast interconnect (thin-fast). The wide-slow configuration approximates a cache using conventional RC interconnect and the thin-fast configuration approximates a cache using an advanced technology such as on-chip transmission lines. The non-shaded rows of Table 3-2 break down the two configurations' input parameters. Due to the fact CIM ignores many details, the model cannot accurately recreate the high inter-switch contention that exists in the thin-fast network. Therefore, in order to reproduce the thinner network's increased bandwidth contention, the parallel links parameter reduces from 8 links for the wide-slow configuration to 3 links for the thin-fast configuration. Though thin-fast links provide single cycle latency, messages occupy the links for 5 cycles because of their limited bandwidth. In comparison, wide-slow messages traverse 4 link-switch pairs on average, but only occupy the links for a single cycle.

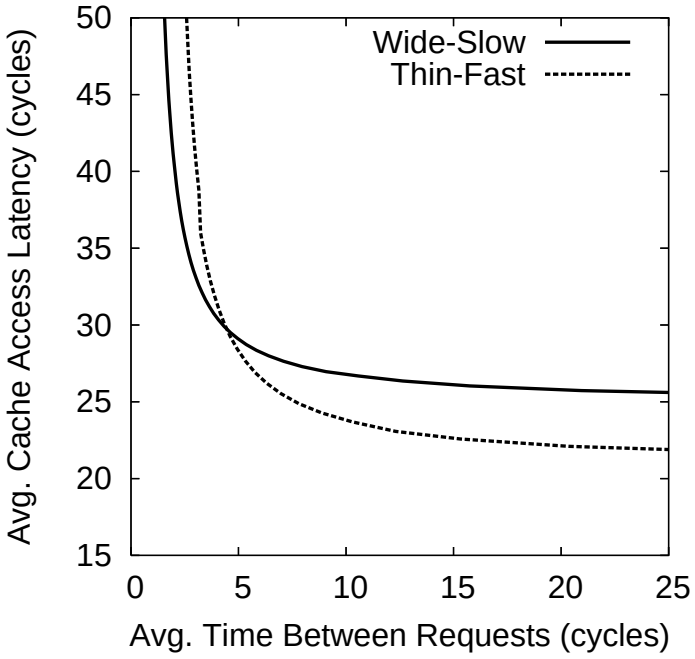


FIGURE 3-4. CIM: Wire Technology - Cache Access Time vs. Bandwidth Demand

CIM's investigation of wire technology indicates advanced wire technology will improve cache performance until bandwidth demand exceeds a certain threshold. Figure 3-4 plots the average cache access latency of both the wide-slow and thin-fast cache configurations versus bandwidth demand. While the wide-slow configuration provides a consistent 26 cycle access latency until the average time between processor requests reduces below 10 cycles, the thin-fast configuration attains access latency as low as 22 cycles, but its latency significantly increases as the average time between requests shrinks below 15 cycles. Therefore, under periods of low bandwidth demand, the thin-fast configuration will outperform the wide-slow configuration. However, high bandwidth demand diminishes thin-fast's performance advantage, and extreme bandwidth demand causes the thin-fast configuration to perform worse than the wide-slow configuration.

Replacing wide and slow wires with thinner and faster wires is not the only way to improve CMP cache access latency. The next section discusses how CMP cache organization can also impact performance.

3.3 Cache Organization and Memory System Performance

This section focuses on how CMP cache organization impacts memory system performance. Previously, Section 3.2 showed moderate bandwidth demand (> 10 cycles between processor requests) barely impacted cache access latency for highly partitioned caches (≥ 64 banks) utilizing wide links. Consequently, when modelling caches using thin and fast links, first-order memory system performance equations provide insight by just considering latency. This section relates average cache access latency to overall memory system performance for both a shared and private CMP cache design. The consequences of these models illustrate how variations in cache latencies and usage affect memory system performance.

3.3.1 Shared CMP Cache

The shared CMP cache organization [32, 62] assumes each processor has private L1 caches, while the on-chip L2 cache is shared among all processors. Ignoring cache coherence details, requests that miss in the local L1 cache are sent to the shared L2 cache where the request could hit, be forwarded to remote L1 caches, or be sent off-chip. For a CMP utilizing a shared cache, memory requests split into four categories: local L1 cache hits, remote L1 hits, L2 cache hits, and off-chip misses. Equation 3.8 presents the normalized memory cycles per instruction for a shared CMP cache organization. In Equation 3.8, P_x is the probability of a memory request being satisfied by the entity x, where x is a local L1 cache, a remote L1 cache, the shared L2 cache, or main memory and L_x equals the latency of each entity.

$$\frac{\text{Shared Cache Memory Cycles}}{\text{Instruction}} = \frac{(P_{localL1} \times L_{localL1}) + (P_{remoteL1} \times L_{remoteL1}) + (P_{L2} \times L_{L2}) + (P_{miss} \times L_{miss})}{Instructions} \tag{3.8}$$

Assuming uniform utilization of L2 cache banks, L_{L2} equals the average L2 cache access latency.

3.3.2 Private CMP Caches

The private CMP cache organization assumes each processor has both private L1 and L2 caches. Requests that miss in the local L1 cache are sent to the local L2 cache where the request could hit, be forwarded to remote L1 and L2 caches, or be sent off-chip. For a CMP utilizing private L2 caches, memory requests split into five categories: local L1 cache hits, remote L1 hits, local L2 cache hits, remote L2 cache hits, and off-chip misses. Equation 3.9 presents the normalized memory cycles per instruction for this private CMP cache organization. Similar to Equation 3.8, P_x in Equation 3.9 is the probability of a memory request being satisfied by the entity *x*, where *x* is a local L1 cache, a remote L1 cache, a local L2 cache, a remote L2 cache, or main memory and L_x equals the latency of each entity.

$$\frac{\text{Private Cache Memory Cycles}}{\text{Instruction}} = \frac{(P_{localL1} \times L_{localL1}) + (P_{remoteL1} \times L_{remoteL1})}{Instructions} + \frac{(P_{localL2} \times L_{localL2}) + (P_{remoteL2} \times L_{remoteL2}) + (P_{miss} \times L_{miss})}{Instructions} \quad (3.9)$$

In general, when comparing the shared cache equation (Equation 3.8) with the private cache equation (Equation 3.9), one can suspect similar L1 cache terms, but Equation 3.8's L_{L2} term will be approximately 2x Equation 3.9's $L_{localL2}$ term and only slightly less than Equation 3.9's $L_{remoteL2}$ term. However, to compensate for it's slower latency, the shared cache can expect Equation 3.8's P_{L2} term to be roughly 1.2x the sum of Equation 3.9's $P_{localL2}$ and $P_{remoteL2}$ terms depending on the workload.

Next, Chapter 4 discusses how migration and replication can improve the relationship between $P_{localL2}$, $P_{remoteL2}$, and P_{miss} by exploiting workload behavior.

3.4 Summary

This chapter illustrated how wire technology and cache organization affects memory system performance. The chapter showed cache partitioning reduces local wire delay's impact on cache bank access latency and improves cache bandwidth. However, as cache partitioning increases, global wire delay between cache banks becomes a more dominant performance bottleneck. Advances in wire technology will reduce global wire latency, but high bandwidth demands will diminish their latency advantage. A different approach to managing wire latency relies on exploiting workload behavior via migration and replication. The next chapter discusses how migration and replication reduces latency with both a shared and private CMP cache hierarchy.

Chapter 4

Exploiting Workload Behavior

This chapter demonstrates the potential of exploiting workload behavior to reduce on-chip wire delay. First, Section 4.1 characterizes commercial and scientific workload behavior for an 8-processor CMP. Then, Section 4.2 and Section 4.3 present analytical models describing how cache block migration and replication impact memory system performance. These abstract models provide high-level understanding of the interaction between workload behavior and migration/replication. For instance, the migration model shows that inter-processor sharing limits migration's benefit, and the replication model demonstrates that selectively replicating frequently requested blocks provides better performance than naively replicating all shared blocks. Furthermore, the replication model illustrates that cache capacity and memory latency significantly impact the optimal amount of replication.

4.1 Characterizing Sharing Types

In order to investigate the performance impact of migration and replication in a CMP cache, this section focuses on understanding workload behavior. Specifically, this section examines an eight SPARC V9 processor CMP using the full system simulator Simics [72] extended with the GEMS memory system timing model [75]. Each processor's L1 caches are split and store 64 KB with 4-way set associativity. Running the Solaris 9 operating system, both commercial and scientific workloads are evaluated. Table 4-1 describes the four studied commercial workloads. The four scientific workloads include two SpecOMP benchmarks [6]: Apsi and Art and two Splash2 benchmarks [115]: Barnes (128k-particles) and Ocean (514×514). To address multithreaded workload variability [3], all workload evaluations use a work-related throughput

TABLE 4-1. Workload Descriptions

Static Web Serving: Apache. We use Apache 2.0.43 for SPARC/Solaris 9, configured to use pthread locks and minimal logging as the web server. We use SURGE [11] to generate web requests. We use a repository of 20,000 files (totalling ~500 MB), and disable Apache logging for high performance. We simulate 3200 clients each with 25 ms think time between requests, and warm-up for ~2 million requests.
Java Server Workload: SPECjbb. SPECjbb2000 is a server-side java benchmark that models a 3-tier system, focusing on the middleware server business logic. We use Sun's HotSpot 1.4.0 Server JVM. Our experiments use 1.5 threads and 1.5 warehouses per processor, a warm-up interval of 200,000 transactions, a data size of ~44 MB.
Online Transaction Processing (OLTP): DB2 with a TPC-C-like workload. The TPC-C benchmark models the database activity of a wholesale supplier, with many concurrent users performing transactions. Our OLTP workload is based on the TPC-C v3.0 benchmark using IBM's DB2 v7.2 EEE database management system. We use a 5 GB database with 25,000 warehouses stored on eight raw disks and an additional dedicated database log disk. We reduced the number of districts per warehouse, items per warehouse, and customers per district to allow more concurrency provided by a larger number of warehouses. There are 128 simulated users, and the database is warmed up for 100,000 transactions before taking measurements.
Static Web Serving: Zeus. Zeus is another static web serving workload driven by SURGE. Zeus uses an event-driving server model. Each processor of the system is bound by a Zeus process, which is waiting for web serving event (e.g., open socket, read file, send file, close socket, etc.). The rest of the configuration is the same as Apache (20,000 files of ~500 MB total size, 3200 clients, 25 ms think time, ~2 million requests for warm-up).

TABLE 4-2. Evaluation Methodology

Benchmark	Fast Forward	Warm-up	Executed
Commercial Workloads (unit = transactions)			
apache	500000	2000	1000
jbb	1000000	15000	10000
oltp	100000	300	200
zeus	500000	2000	2000
Scientific Workloads (unit = billion instructions)			
apsi	89	4.6	loop completion
art	121	3.2	loop completion
barnes	17.3	1.9	run completion
ocean	None	2.4	run completion

metric. Thus for the commercial workloads, run lengths equal transactions completed and for the Splash2 workloads, runs completed after the warm-up period indicated in Table 4-2. 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. Thus throughput metrics, rather than

TABLE 4-3. Percentage of Cache Blocks Profiled at L2 Eviction

Benchmark	% Profiled at L2 Eviction
apache	94%
jbb	82
oltp	85
zeus	92
apsi	81
art	99
barnes	61
ocean	97

IPC, measure workload performance. Finally, the workload characterization data presented in this chapter is based on five runs of each workload.

The workload 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. To isolate the sharing activity from access latency, the simulation model assumes a single-banked 16 MB inclusive shared L2 cache with 16-way associativity and a uniform access time. To mitigate cold start effects, all workloads except Barnes run long enough so that L2 cache misses outnumber physical L2 cache blocks by at least 4 times. Therefore, most blocks are profiled at the completion of their *L2 cache lifetimes*, i.e. when they are evicted from the L2 cache, but some blocks have very long L2 cache lifetimes and are profiled when the simulation completes. Table 4-3 shows the percentage of L2 blocks profiled when they are evicted from the L2 cache.

The cost and benefit of migration and replication depend on the cache block's sharing behavior. This dissertation identifies 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. Assuming a private CMP cache using a broadcast protocol, single requestor blocks benefit from migration, but cannot benefit from replica-

TABLE 4-4. L2 Cache Request Profile

Bench	Single Requestor			Shared Read-Only			Shared Read-Write		
	% of Requests	Avg. Active Time (M cycles)	Avg. Requests / Block	% of Requests	Avg. Active Time (M cycles)	Avg. Requests / Block	% of Requests	Avg. Active Time (M cycles)	Avg. Requests / Block
apache	13%	> 1	2	44%	8	44	43%	5	12
jbb	57	2	4	42	12	42	1	9	32
oltp	4	1	2	71	11	104	25	7	18
zeus	20	> 1	2	55	6	64	26	3	8
apsi	> 99	2	4	< 1	1	9	< 1	2	39
art	56	4	14	44	8	28	< 1	13	22
barnes	20	30	7	74	120	330	7	87	8
ocean	94	7	5	1	3	11	5	15	47

tion. Shared read-only and shared read-write blocks can benefit from both, but replicating shared read-write data will incur extra delay on writes due to coherence invalidations.

Within a CMP cache, each of these three sharing types exhibit distinct behavior. The following subsections begin by identifying the request, capacity, and sharing behavior of the three sharing types. Then, in order to provide the details required by the models in Section 4.2 and Section 4.3, the section concludes by characterizing the three sharing type's request locality, working set size, and probability distribution functions.

4.1.1 Requests

To understand the usage of L2 cache blocks, this subsection analyzes the requests satisfied by each sharing type. The subsection shows the percentage of requests to each sharing type varies significantly between the workloads. Table 4-4 shows that shared read-only requests dominate the four commercial workloads (42-71% of requests), while Art and Barnes (44% and 74% respectively) are the only scientific workloads to make more than 1% of its requests to shared read-only blocks. Additionally, the commercial workloads Apache, Oltp, and Zeus, issue many requests to shared read-write blocks (25-43%), while the other five

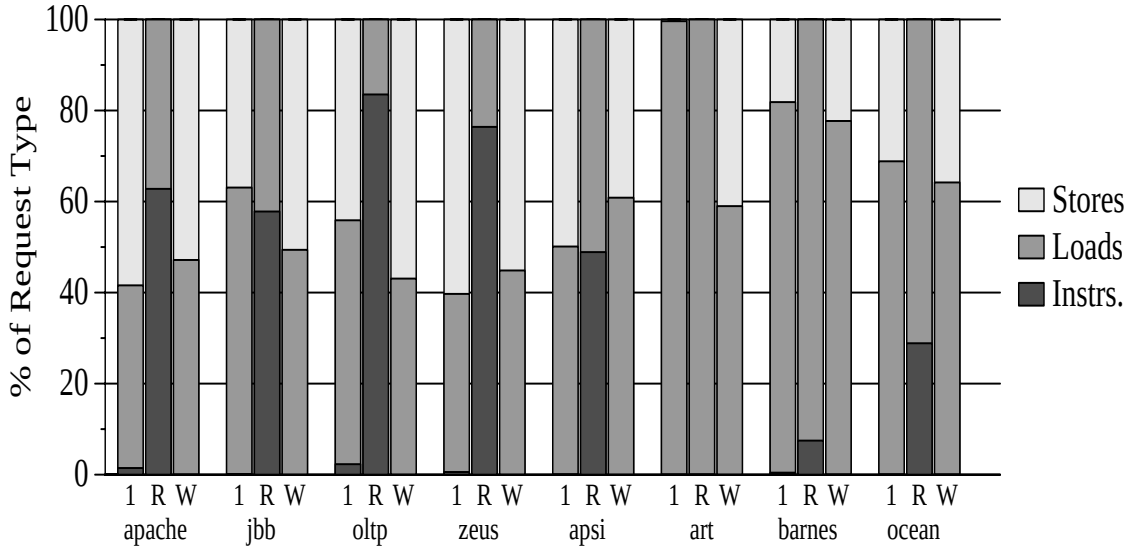


FIGURE 4-1. L2 Cache Shared Requests Breakdown
(1: Single Requestor, RO: Shared Read-only, RW: Shared Read-write)

workloads issue no more than 7% of shared read-write requests. Finally, single-requestor blocks account for nearly all the requests by Apsi and Ocean, nearly half for Jbb and Art, and relatively little for the rest.

The high percentage of shared read-only requests causes shared read-only blocks to have longer active lifetimes and higher average utilizations than the other block types. In particular, Table 4-4‘s average active time columns display the average number of cycles between the first and last request to a particular L2 block. Meanwhile, Table 4-4‘s average requests per block columns show the average number of request to a particular L2 block. Across practically all workloads, shared read-only blocks exhibit the longest average active lifetimes and highest average requests, while shared read-write blocks rank between shared read-only blocks and single requestor blocks. Though single requestor blocks satisfy a majority of requests in certain workloads, they average only 2-14 requests during their on-chip lifetimes.

In terms of L2 request types, the commercial and scientific workloads exhibit distinct behavior. For commercial workloads, Figure 4-1 indicates that single requestor requests roughly evenly split between loads and stores. In contrast, for all scientific workloads except Apsi, single requestor requests highly bias towards loads. For the commercial workloads, instruction fetches contribute between 58% to 84% of

TABLE 4-5. L2 Cache Capacity and Allocation Profile

Benchmark	Single Requestor		Shared Read-Only		Shared Read-Write	
	Avg. % of Capacity	% of Allocations	Avg. % of Capacity	% of Allocations	Avg. % of Capacity	% of Allocations
apache	51%	63%	21%	8%	28%	29%
jbb	91	93	9	7	< 1	< 1
oltp	53	48	20	18	27	35
zeus	72	76	11	5	16	19
apsi	> 99	> 99	< 1	< 1	< 1	< 1
art	77	74	24	26	< 1	< 1
barnes	92	73	4	6	3	22
ocean	99	99	< 1	< 1	1	1

shared read-only requests, while for the scientific workloads, loads account for the majority of shared read-only requests. Finally, for the commercial workloads, shared read-write requests exhibit an even split between loads and stores, while for the scientific workloads, loads contribute at least 59% of all shared read-write requests. By combining the data of Table 4-4 with Figure 4-1, one observes CMP caches must manage commercial workload's large instruction footprints to improve performance, while scientific workload performance improvement more closely depends on reducing load latency.

4.1.2 Cache Capacity

To understand the cache pressure placed on the L2 cache, this subsection analyzes the cache capacity consumed by each sharing type. The subsection demonstrates that though shared data dominates requests, single-requestor blocks consume the majority of the cache capacity. Table 4-5 shows that single-requestor blocks account for over 50% of average 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 50%. Table 4-5 also presents the percentage of block allocations for each sharing type. While the average percentage of capacity indicates cache storage distribution, the percentage of allocations correlates to the number of off-chip requests for each sharing type. By com-

paring the capacity and allocations columns, one observes allocations match or exceed consumed capacity for all sharing types except shared read-only data in commercial workloads. Instead, for these commercial workloads, shared read-only data allocations are significantly lower relative to their consumed capacity. This reduction is due to the fact that shared read-only L2 block lifetimes are twice as long as single requestor blocks and at least 20% longer than shared read-write blocks. Replicating shared blocks in private caches to reduce access latency is attractive, since they are accessed frequently and have long L2 cache lifetimes yet consume relatively little cache capacity.

4.1.3 Sharing Behavior

While replicating shared read-only data is attractive, blind replication is dangerous, since the degree of sharing suggests that the capacity could increase significantly. Table 4-6 shows the probability that the next processor to request a L2 block usually differs from the last requesting processor. This behavior is especially true for shared read-only blocks where the different requestor probability exceeds 0.83 for all workloads except Barnes. Therefore, replicating rather than migrating shared read-only data close to the multiple requesting processors appears advantageous. However, the high average number of sharers for shared read-only data indicates allowing all replication will significantly reduce the effective cache capacity. Specifically, 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 lifetime. Fully replicating these blocks could increase the effective working set by 25-74%. Therefore, for shared read-only blocks, CMP caches must balance replication's on-chip latency reduction benefit with replication's decrease of effective cache capacity.

In comparison, replicating shared read-write data is less advantageous. Allowing multiple shared read-write copies permits faster read latency by moving shared copies close to multiple processors. In contrast, storing a single shared read-write block facilitates quicker cache-to-cache transfers by reducing costly coherence invalidations. Table 4-6 displays shared read-write blocks are shared less widely than shared

TABLE 4-6. L2 Cache Block Sharing Behavior

Benchmark	Shared Read-Only		Shared Read-Write		
	Different Requestor Prob.	Avg. # of Sharers	Different Requestor Prob.	Avg. # of Sharers	Run Length
apache	.90	3.6	.61	2.8	1.2
jbb	.91	3.4	.62	2.4	1.1
oltp	.83	4.5	.53	3.6	1.4
zeus	.88	3.0	.56	2.3	1.3
apsi	.91	7.2	.58	2.7	1.5
art	.85	3.0	.25	2.3	3.1
barnes	.66	3.2	.41	2.1	4.2
ocean	.83	4.7	.36	2.1	4.6

read-only blocks. However, for the commercial workloads, shared read-write blocks are still shared frequently enough that their next requestor is likely not to be their last requestor. Also, Table 4-6 presents the average run lengths of these blocks. Inspired by Eggers and Katz write run characterization [34], the average run length is the average combination of reads to a L2 block between writes from different processors and remote reads to a L2 block between writes from the same processor. For the three commercial workloads that issue many shared read-write requests—apache, oltp, and zeus—, only 1.1-1.3 intervening requests occur between writes. Thus, for the evaluated workloads, allowing shared read-write data replication appears less advantageous.

4.1.4 Request vs. Cache Block Locality

Fortunately, shared read-only blocks exhibit strong locality, especially for commercial workloads, while single requestor and shared read-write blocks demonstrate less locality. Figure 4-2 plots the cumulative distribution of single requestor requests across the cumulative percent of single requestor blocks. Interestingly, three of the four commercial workloads exhibit little locality for single requestor data. Specifically, the top 20% of single requestor blocks in OLTP, Apache, and Zeus account for less than 60% of the single requestor requests. The scientific workloads demonstrate more locality, with the top 20% of single

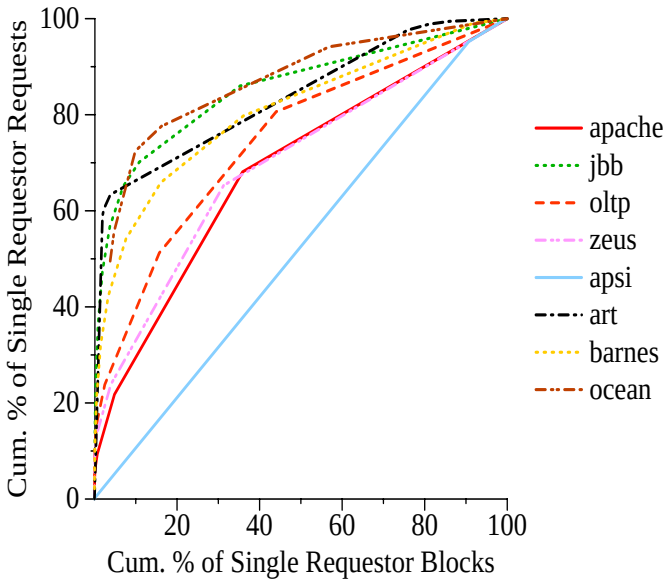


FIGURE 4-2. Request to Block Distribution: Single Requestor Data

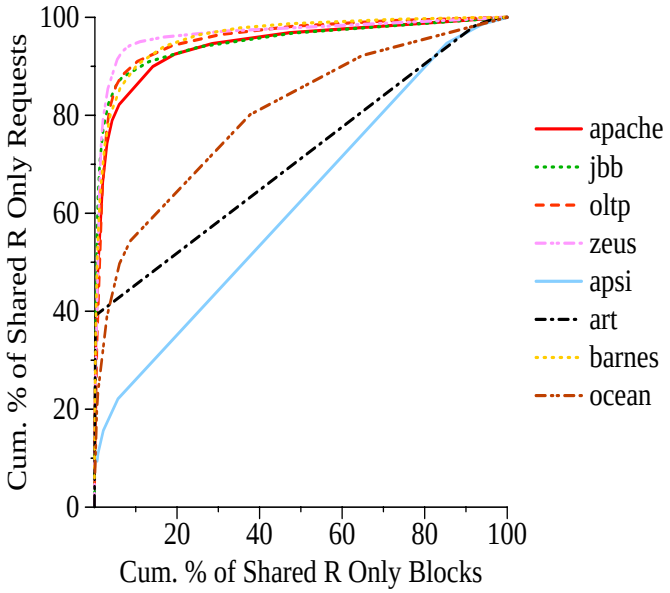


FIGURE 4-3. Request to Block Distribution: Shared Read-only Data

requestor blocks in Art, Barnes, and Ocean satisfying 55-80% of single requestor requests. However, further observation reveals the data footprint of the top 20% of single requestor blocks in Art, Barnes, and Ocean is at least 3 MB.

In contrast, shared read-only data displays tremendous locality in the commercial workloads. Figure 4-3 plots the cumulative percent distribution of shared read-only requests across the cumulative percent distri-

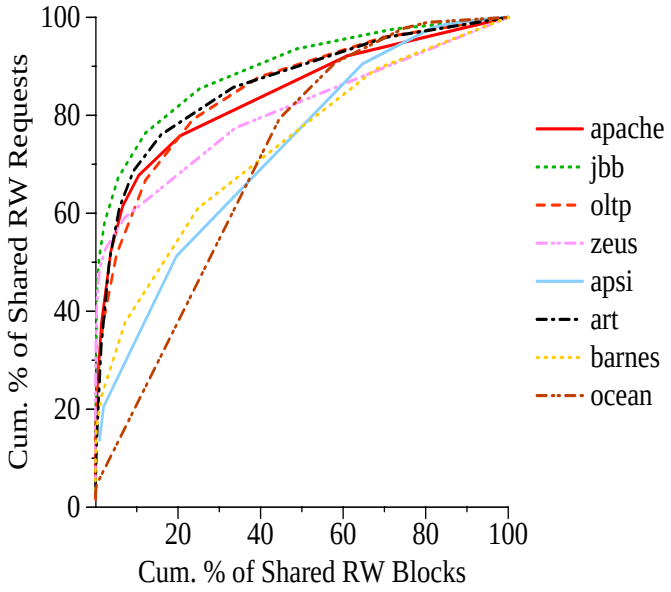


FIGURE 4-4. Request to Block Distribution: Shared Read-write Data

bution of 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. Further observation reveals the data footprint of the top 3% of shared read-only blocks in JBB, OLTP, and Zeus is only 100-300 KB. Due to their small footprints, selectively replicating only the most frequently requested blocks can reduce shared read-only request latency without risking significantly decreasing effective cache capacity.

Finally, Figure 4-4 reveals shared read-write data have locality characteristics between single requestor data and shared read-only data. The top 3% of shared read-write blocks (1 KB-1 MB) account for 20-60% of shared read-write requests. Therefore, selectively replicating shared read-write data is less advantageous than selectively replicating shared read-only data. Also, as previously discussed in Section 4.1.3, the lack of intervening reads between writes does not justify replicating shared read-write data.

Because shared read-only data exhibit tremendous locality and don’t suffer coherence invalidations, this dissertation focuses on selectively replicating shared read-only blocks.

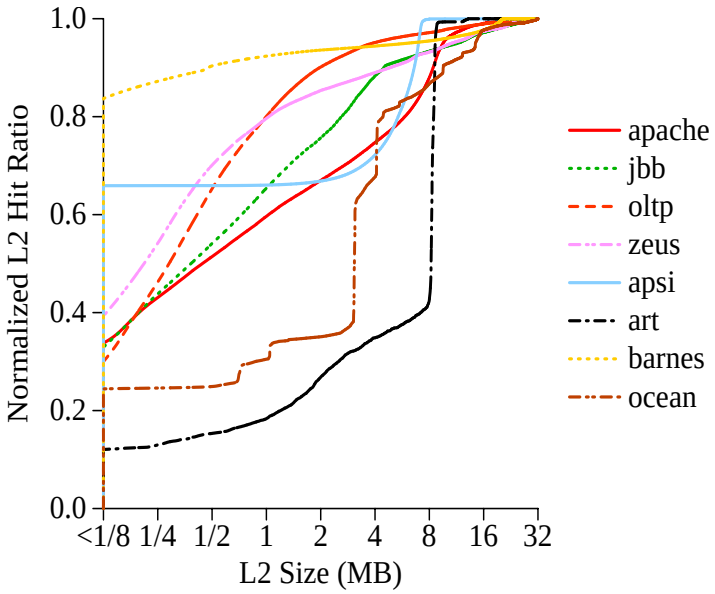


FIGURE 4-5. Normalized L2 Cache Hit Ratios

4.1.5 Cache Hit Ratio

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 4-5 illustrates this risk by plotting the normalized hit ratios for fully-associative caches up to 32 MB, Equation 4.1.

$$\text{Normalized L2 Cache Hit Ratio} = \frac{\text{Hits within cache size } \alpha}{\text{Hits within a 32 MB L2 cache}} \quad (4.1)$$

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, Ocean and Art have critical working set sizes of 4 MB and 8 MB, respectively. Increasing the available cache capacity above those thresholds has a dramatic negative impact on performance. All of the scientific workloads exhibit clearly identifiable working set boundaries, while the commercial workloads

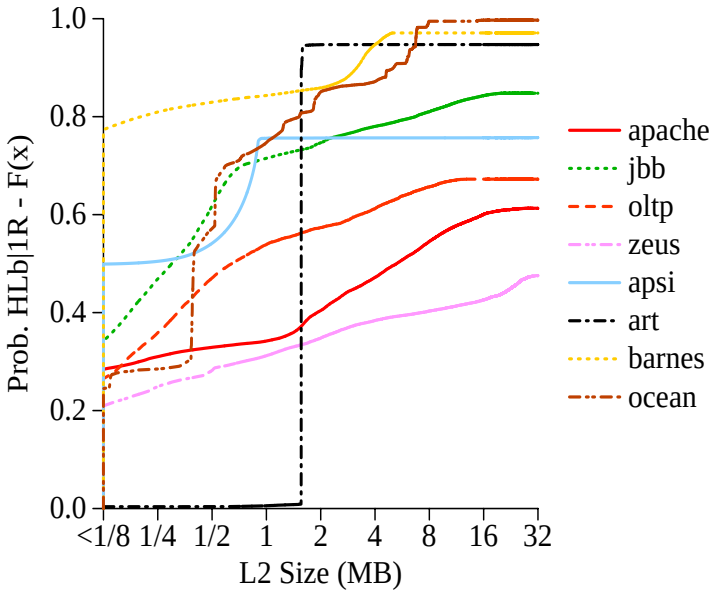


FIGURE 4-6. Probability Distribution Function HLb|1R - F(x)

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

4.1.6 Probability Distribution Functions

This subsection presents the probability distribution functions required by both the migration and replication analytic models. Similar to the previous normalized cache hit ratio, Probability Distribution Functions (PDFs) illustrate the probability a request will hit in a given sized cache. Furthermore, conditional PDFs can determine the hit probability for a certain type of request. In particular, the analytical models of the following sections require discrete conditional PDFs to determine the hit rate for certain request types. These PDF plots illustrate the hits to both the local and remote LRU stacks. Interestingly, the general shapes of the local hit curves for the scientific workloads exhibit noticeable steps, while the other curves increase gradually.

Section 4.2‘s migration model requires three discrete conditional probability distribution functions: the probability of a local hit given the request is for a single requestor block—$P_{HLb|1R}$—(Figure 4-6), the

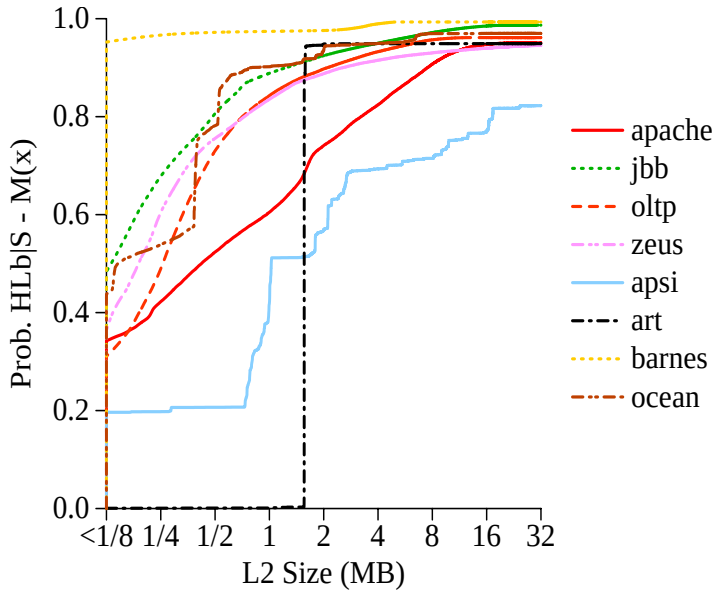


FIGURE 4-7. Probability Distribution Function HLb|S - M(x)

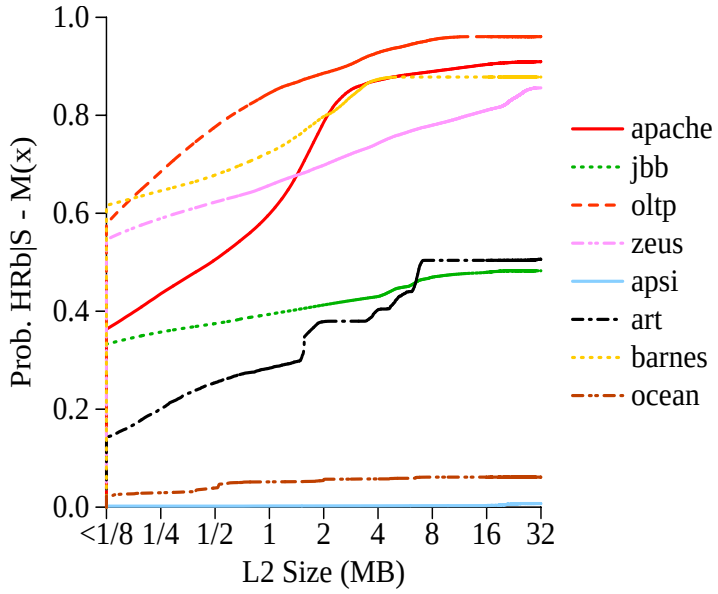


FIGURE 4-8. Probability Distribution Function HRb|S - N(x)

probability of a local hit given the request is for a shared block—$P_{HLb|S}$—(Figure 4-7), and the probability of a remote hit given the request is for a shared block—$P_{HRb|S}$—(Figure 4-8).

Meanwhile, Section 4.3‘s replication model requires five discrete conditional probability distribution functions: the probability of a local hit given the request is for a single requestor block—$P_{HLb|1R}$—(Figure 4-6), the probability of a local hit given the request is for a shared read-only block—$P_{HLb|SRO}$—(Figure 4-9),

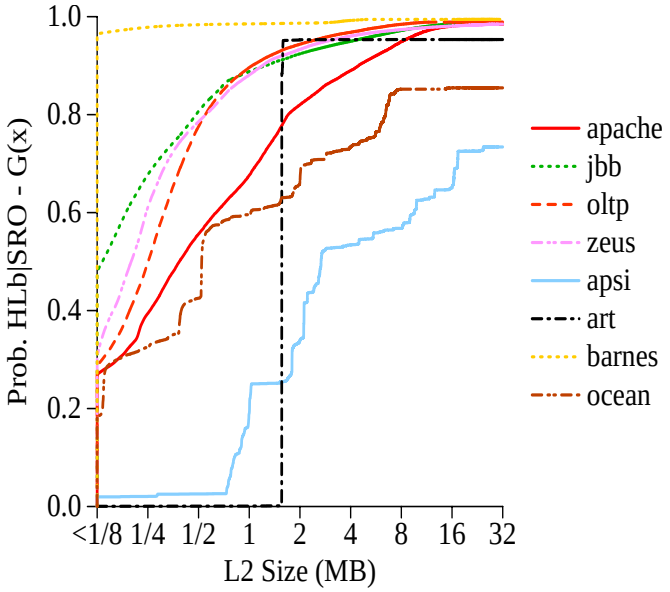


FIGURE 4-9. Probability Distribution Function HLb|SRO - G(x)

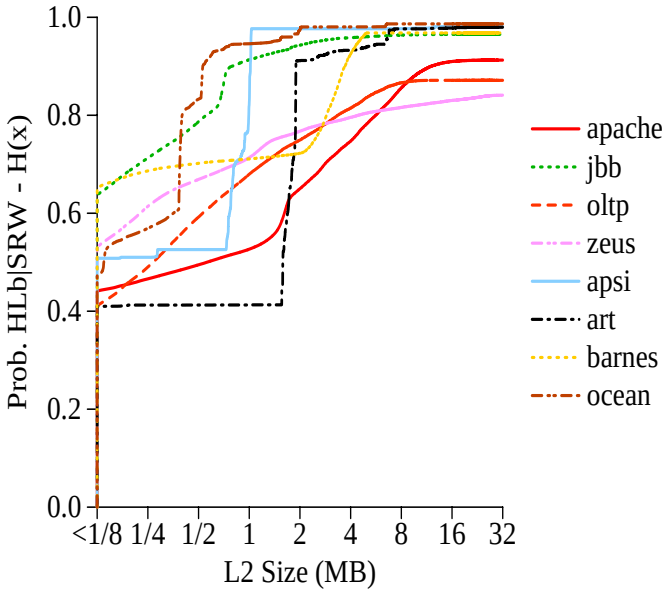


FIGURE 4-10. Probability Distribution Function HLb|SRW - H(x)

the probability of a local hit given the request is for a shared read-only block—$P_{HLb|SRW}$—(Figure 4-10), the probability of a local hit given the request is for a shared read-only block—$P_{HRb|SRO}$—(Figure 4-11),and the probability of a remote hit given the request is for a shared block—$P_{HRb|SRW}$—(Figure 4-12). However, one should note one key difference between the empirical probability distributions presented here and a true distribution function [5]. These empirical probability distributions are limited by finite sim-

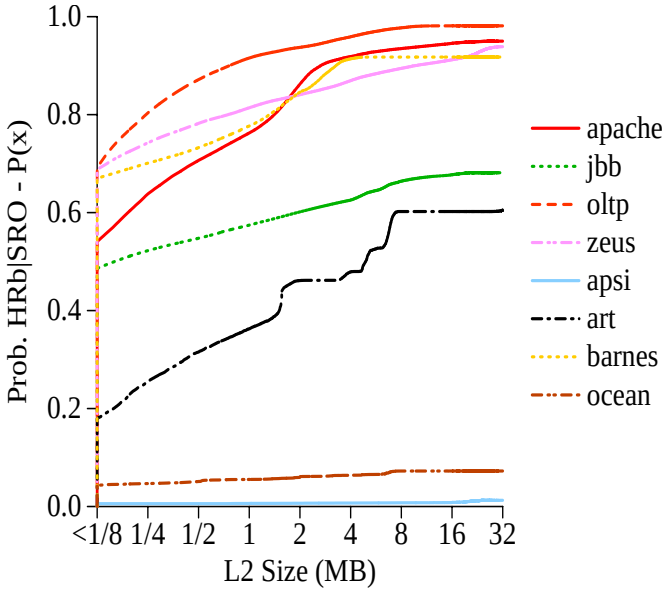


FIGURE 4-11. Probability Distribution Function HRb|SRO - P(x)

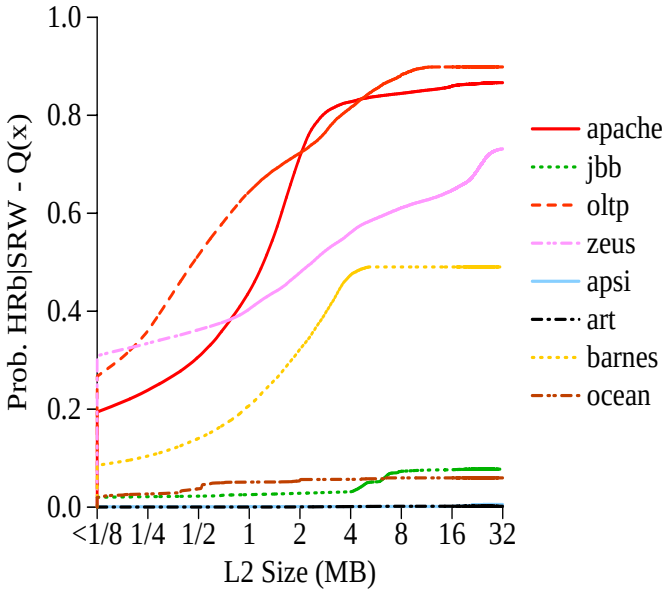


FIGURE 4-12. Probability Distribution Function HRb|SRW - Q(x)

ulation times and memory capacity, and thus cannot approach 1 as cache size increases to infinity. Instead, the probability distribution functions presented in this section stop at the 32 MB. Those hits beyond 32 MB are assumed to be compulsory misses [45].

In order to better understand the benefit of both migration and replication within a CMP cache, the next two sections (Section 4.2 and Section 4.3) present and evaluate two different analytical models using the data presented in this section.

4.2 Exploiting Workload Behavior Through Migration

This section illustrates the potential performance impact of migration in a CMP cache. By moving L2 cache blocks closer to the requesting processors, migration removes on-chip wire delay and improves memory system performance. Previously, Section 3.3 introduced Equation 3.8 and Equation 3.9 to describe the memory system performance of a shared CMP cache and a private CMP cache, respectively. Neglecting cache coherence optimizations, a statically shared CMP cache allows no migration and a private CMP cache can implement various migration policies. In particular, one easy to examine migration policy moves L2 cache blocks directly to the last requesting processor's private L2 cache with the swapped out block moving into the migrated block's previous location. Other migration policies exist, for both shared and private caches, but cannot be easily modeled analytically and instead require execution driven simulation. Therefore, further discussion of these more complicated policies is delayed until Chapter 5. The remainder of this section focuses on understanding direct migration's benefit in a private CMP cache. Section 4.2.1 proposes a model for the direct migration policy, then Section 4.2.2 illustrates that the performance benefit of direct migration depends on the probability of the requesting processor being the last requestor.

4.2.1 Modelling Migration

This section constructs an abstract model to analyze the high-level benefits and limitations of migration. Specifically, the model assumes an 8-processor private CMP cache with each processor having fast access to its local L2 cache bank and slower access to the remote L2 cache banks. For simplification reasons, the

model assumes migration has no effect on the L2 miss rate and all caches are fully associative with perfect LRU replacement. Also when a block is migrated from one cache bank to another, not only is the migrated block assumed to be within both the previous owner processor's and the current owner processor's local MRU stacks, but also the subsequent swapped block is assumed to be within both MRU stacks. The result is the effective on-chip cache size (*b*) can be greater than the local private L2 cache size because some shared blocks may exist in other processor's local cache banks. In particular, the shared blocks last requested by other processors are assumed to be in remote on-chip cache banks. Therefore, based on the previous Equation 3.9, $P_{localL2}$ equals the probability a requesting processor was the last processor to request a given cache block that lies within the local Most Recently Used (MRU) stack of size *b:*

$$P_{localL2} = P_{LR|HLb} = \text{prob. last requestor given window local MRU stack size b}$$

$$= P_{1R} \times P_{HLb|1R} + P_S \times P_{LR|S} \times P_{HLb|S} \tag{4.2}$$

where:

$$P_{1R} = P_{SingleRequestor} = \text{prob. of a request being for single requestor data} \tag{4.3}$$

$$P_S = P_{Shared} = \text{prob. of a request being for shared data} \tag{4.4}$$

$$P_{LR|S} = \text{probability of Last Requestor given S} \tag{4.5}$$

$$P_{HLb|1R} = F(0 \le X \le b) = \sum_{0}^{b} f(x) = \text{prob. of first requestor hit in local MRU stack of size b} \quad (4.6)$$

$$P_{HLb|S} = M(0 \le X \le b) = \sum_{0}^{b} m(x) = \text{prob. of shared hit in local MRU stack of size b} \quad (4.7)$$

$$b = \frac{\text{Private Cache Size}}{(1-(P_S \times P_{\overline{LR}|S}))} \quad (4.8)$$

Similarly, $P_{remoteL2}$ equals the probability a requesting processor was the last requesting processor given the shared cache block is within the local MRU size of *b*:

$$P_{remoteL2} = P_S \times P_{\overline{LR}|S} \times P_{HRb|S} = \text{prob. not last requestor \& hits in remote MRU stack size b} \quad (4.9)$$

where:

$$P_{HRb|S} = N(0 \le X \le b) = \sum_{0}^{b} n(x)dx = \text{prob. of S hit in 7 remote MRU stacks of size b} \quad (4.10)$$

Finally, P_{miss} equals the probability a request misses both in the local L2 cache and the remote L2 caches:

$$P_{miss} = 1-(P_{1R} \times P_{HLb|1R} + P_S \times P_{LR|S} \times P_{HLb|S}) - P_S \times P_{\overline{LR}|S} \times P_{HRb|S} \quad (4.11)$$

In order to isolate migration's effect on L2 cache access latency, Equation 4.12 drops the L1 cache terms of Equation 3.9. The result is the following memory system performance equation for L1 misses:

$$\frac{\text{Private Cache L1 Miss Cycles}}{\text{Instruction}} = \frac{(P_{localL2} \times L_{localL2}) + (P_{remoteL2} \times L_{remoteL2}) + (P_{miss} \times L_{miss})}{(Instructions/L1misses)} \quad (4.12)$$

In comparison, Equation 4.13 presents the memory system performance equation for L1 misses assuming a shared CMP cache that disallows migration:

$$\frac{\text{Shared Cache L1 Miss Cycles}}{\text{Instruction}} = \frac{(P_{sharedL2} \times L_{sharedL2}) + (P_{miss} \times L_{miss})}{(Instructions/L1misses)} \quad (4.13)$$

where:

$$P_{sharedL2} = P_{AggregateStackHit} = \text{prob. of hit in aggregate MRU stack of size} = \text{total L2 capacity} \quad (4.14)$$

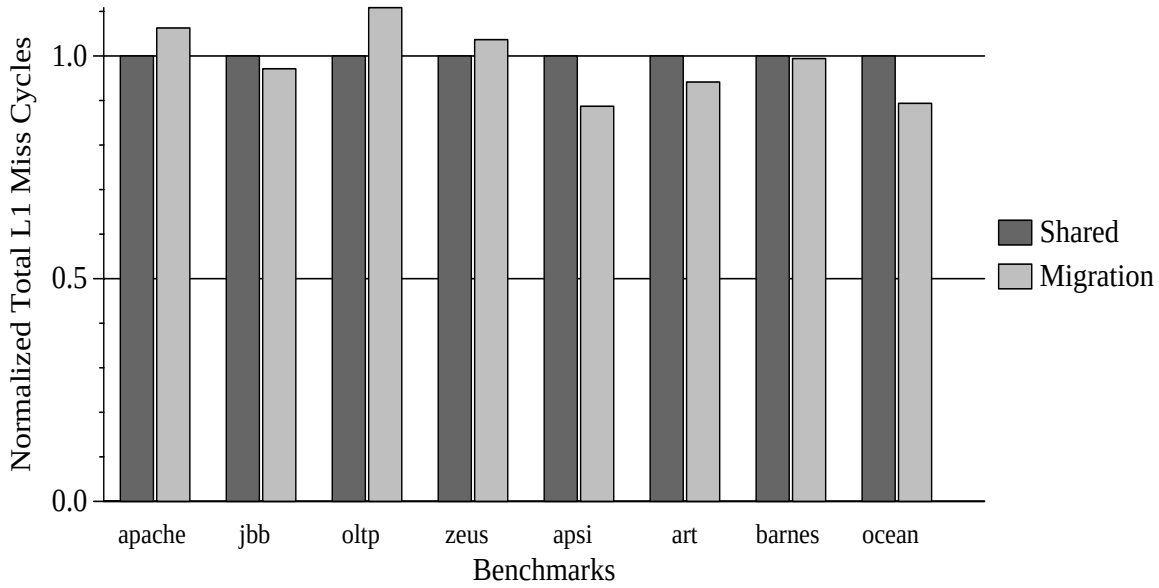


FIGURE 4-13. Migration Model: *All Workloads* Default

4.2.2 Evaluating Migration

This section demonstrates migration's performance improvement highly depends on the sharing behavior. The evaluated model is not accurate enough to project absolute performance, but does provide high-level understanding of migration's benefits and limitations. The model uses the empirical data provided in Section 4.1 and sets $L_{localL2}$ to 20 cycles, $L_{remoteL2}$ to 50 cycles, $L_{sharedL2}$ to 44 cycles, and L_{miss} to 300 cycles. Figure 4-13 plots the total L1 miss cycles of a 8 MB private CMP cache using direct migration (Equation 4.12) normalized to the total L1 miss cycles of a 8 MB shared CMP cache (Equation 4.13). For the commercial workloads, migration reduces Jbb's total L1 miss cycles by 3%, but degrades performance for the other three commercial workloads by as much as 11%. These three commercial workloads encounter performance degradation because the majority of their requests are for shared data (Table 4-4) with high probability a different processor was the last requestor (Table 4-6). Thus migration exacerbates on-chip latency because many shared requests hit in remote cache banks.

In contrast, migration improves performance for all four scientific workloads, with the degree of improvement directly corresponding to the percentage of single requestor requests (Table 4-4). For instance, Apsi and Ocean exhibit the largest percentage of single requestor requests (99% and 94% respectively) and thus they encounter the most significant reduction in total L1 miss cycles (12% and 11% respectively). Meanwhile, Art and Barnes exhibit a smaller percentage of single requestor requests (53% and 19% respectively) and thus they encounter only a small reduction in total L1 miss cycles (6% and 1% respectively).

Overall, the migration model demonstrates the performance improvement provided by direct migration directly corresponds to a workload's sharing behavior. Migration significantly benefits workloads with little temporal sharing, but migration can degrade performance for workloads with a large amount of sharing.

4.3 Exploiting Workload Behavior Through Replication

Similar to the previous migration evaluation, this section focuses on illustrating the potential performance impact of replication in a CMP cache. Analogous to migration, a statically shared CMP cache disallows replication, where a private CMP cache can implement various replication policies. These replication policies range from replicating all shared L2 cache blocks to disallowing replication and instead migrating all L2 cache blocks. Because replicating shared read-write blocks will incur extra delay on writes due to coherence invalidations, the model focuses on replicating shared read-only data and migrating shared read-write and single requestor data.

The degree of shared read-only replication will have a significant impact on CMP cache performance. As demonstrated in Section 4.1, selectively replicating the most frequently requested shared read-only blocks may provide most of replication's benefit without significantly decreasing the effective cache size. The model proposed in Section 4.3.1, evaluates how adjusting the amount of replication impacts memory system performance. Also the model analyzes how CMP cache size and memory latency relate to the performance benefit of selective replication.

4.3.1 Modelling Replication

This section constructs an abstract model to analyze the benefits and limitations of selective replication. The model assumes the same parameters as the model proposed in Section 4.2.1 including the presumption that single requestor, shared read-write, and any non-replicated shared read-only blocks always migrate to the last requestor's L2 cache with the swapped out block lying within the new owning processor's local MRU stack. Since only shared read-only data will be selectively replicated, the model begins (Equation 4.15) by splitting the memory cycles spent on L1 misses in a private CMP cache into the three sharing types:

$$\frac{\text{Private Caches L1 Miss Cycles}}{\text{Instruction}} = \frac{(P_{1R} \times L_{1R}) + (P_{SRO} \times L_{SRO}) + (P_{SRW} \times L_{SRW})}{(Instructions/L1misses)} \quad (4.15)$$

where:

$$P_{1R} = P_{SingleRequestor} = \text{prob. of a request being for single requestor data} \quad (4.16)$$

$$P_{SRO} = P_{SharedReadOnly} = \text{prob. of a request being for shared read-only data} \quad (4.17)$$

$$P_{SRW} = P_{SharedReadWrite} = \text{prob. of a request being for shared read-write data} \quad (4.18)$$

Next, Equation 4.19 introduces the probability of shared read-only block replication:

$$P_{Rep} = P_{Replication} = \text{prob. that a shared read-only block is replicated} \quad (4.19)$$

and Equation 4.20-4.24 present the probability requests of each sharing type hit in a local or remote MRU stack of size b:

$$P_{HLb|1R} = F(0 \le X \le b) = \sum_{0}^{b} f(x) = \text{prob. of first requestor hit in local MRU stack of size b} \quad (4.20)$$

$$P_{HLb|SRO} = G(0 \le X \le b) = \sum_{0}^{b} g(x) = \text{prob. of shared read-only hit in local MRU stack of size b} \quad (4.21)$$

$$P_{HLb|SRW} = H(0 \le X \le b) = \sum_{0}^{b} h(x) = \text{prob. of shared read-write hit in local MRU stack of size b} \quad (4.22)$$

$$P_{HRb|SRO} = P(0 \le X \le b) = \sum_{0}^{b} p(x) = \text{prob. of SRO hit in 7 remote MRU stacks of size b} \quad (4.23)$$

$$P_{HRb|SRW} = Q(0 \le X \le b) = \sum_{0}^{b} q(x) = \text{prob. of SRW hit in 7 remote MRU stacks of size b} \quad (4.24)$$

where b equals the effective private cache size:

$$b = \frac{\text{Private Cache Size}}{(1-(P_{SRW} \times P_{\overline{LR}|SRW} + P_{SRO} \times P_{\overline{LR}|SRO} \times (1-P_{Rep})))} \quad (4.25)$$

To model the accuracy of selectively replicating frequently requested shared read-only blocks, Equation 4.27 introduces the function $locality_{SRO}(x)$, where x is the percentage of L2 cache capacity devoted to shared read-only replicas, *CapRep*. The function returns the probability a requested shared read-only block would have been replicated for the given replication capacity. The function assumes perfect selection of the most frequently requested blocks. Therefore, the value of $locality_{SRO}(x)$ directly corresponds to the shared read-only locality plot in Figure 4-3: (4.26)

$$locality_{SRO}(x) = \text{the locality of SRO hits for x\% capacity for replicas} \tag{4.27}$$

$$CapRep = \text{\% Capacity for Replicas} = P_{SRO} \times P_{\overline{LR}|SRO} \times P_{Rep} \tag{4.28}$$

Equation 4.29 uses $locality_{SRO}(x)$ in combination with $P_{HLb|SRO}$ and $P_{LR|SRO}$ to determine the probability of a local stack hit for a shared read-only request:

$$P_{LSH|SRO} = P_{HLb|SRO} \times (P_{LR|SRO} + P_{\overline{LR}|SRO} \times locality_{SRO}(CapRep)) \tag{4.29}$$

Finally, Equation 4.30 calculates the total cycles spent on L1 misses in a private CMP cache using selective replication:

$$\frac{\text{Private Caches L1 Miss Cycles}}{\text{Instruction}} = \frac{P_{1R} \times (P_{HLb|1R} \times L_{local} + (1 - P_{HLb|1R}) \times L_{miss})}{(Instructions/L1misses)} +$$

$$\frac{P_{SRO} \times P_{LSH|SRO} \times L_{local}}{(Instructions/L1misses)} +$$

$$\frac{P_{SRO} \times ((1 - P_{LSH|SRO}) \times P_{HRb|SRO}) \times L_{remote}}{(Instructions/L1misses)} +$$

$$\frac{P_{SRO} \times ((1 - P_{LSH|SRO}) \times (1 - P_{HRb|SRO})) \times L_{miss}}{(Instructions/L1misses)} +$$

$$\frac{P_{SRW} \times ((P_{LR|SRW} \times P_{HLb|SRW}) \times L_{local} + (P_{SRW} \times (P_{\overline{LR}|SRW} \times P_{HRb|SRW})) \times L_{remote})}{(Instructions/L1misses)} +$$

$$\frac{P_{SRW} \times (1 - P_{LR|SRW} \times P_{HLb|SRW} - P_{\overline{LR}|SRW} \times P_{HRb|SRW}) \times L_{miss}}{(Instructions/L1misses)} \quad (4.30)$$

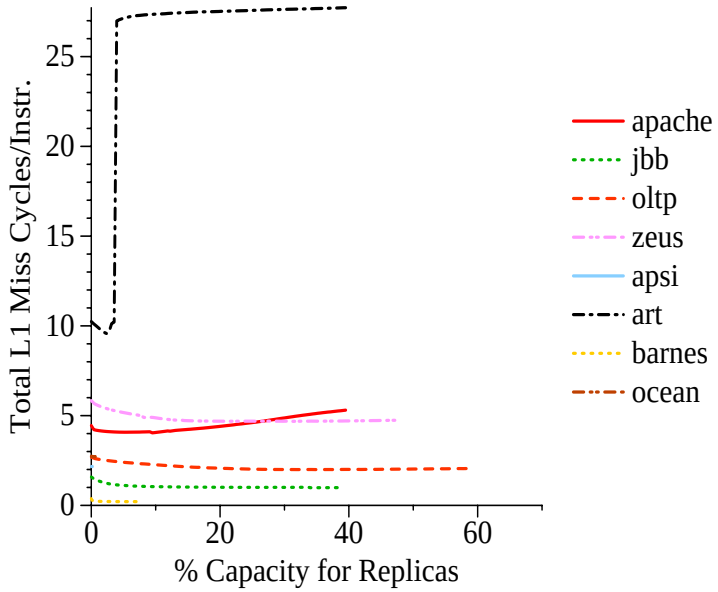


FIGURE 4-14. Replication Model: *All Workloads* Default Parameters

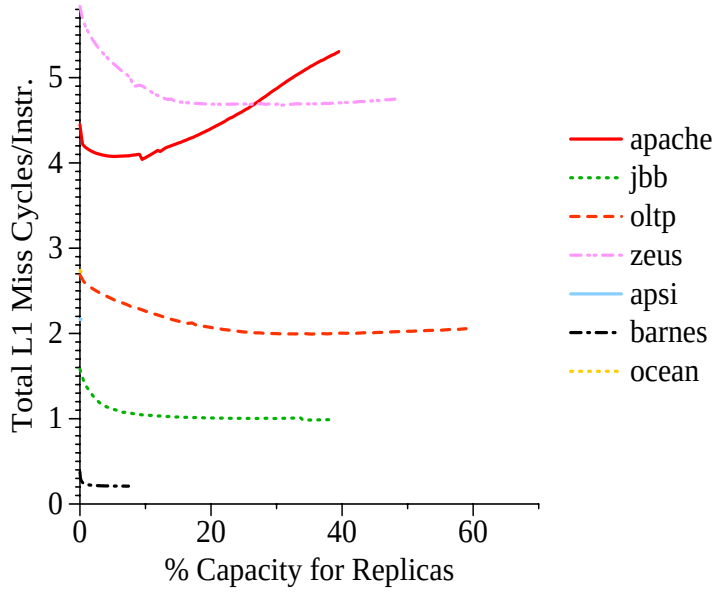


FIGURE 4-15. Replication Model: *All Workloads Except Art* Default Parameters

4.3.2 Evaluating Replication

This section demonstrates the optimal amount of replication depends on workload behavior and system configuration. Similar to the previous migration model, the replication model is not accurate enough to project absolute performance. Instead, the model incorporates enough detail to convey high-level under-

standing of how selective replication impacts performance. Later, using execution driven simulation, Chapter 6 analyzes replication's performance impact for a particular implementation. The replication model uses the data provided in Section 4.1, along with the same input parameters as the migration model. Figure 4-14 plots the total L1 miss cycles of the private CMP cache using selective replication (Equation 4.30) versus the percent of capacity for replicas (Equation 4.28) across the eight evaluated workloads. Because Art's high miss rate skews the y axis of Figure 4-14, Figure 4-15 presents the same data as Figure 4-14 with Art excluded.

For all four commercial workloads, the percentage of capacity devoted to replicas significantly affects performance. For these commercial workloads, Table 4-4 previously showed shared read-only requests accounted for at least 42% of all L2 cache requests. By replicating the most frequently requested shared read-only blocks close to each processor, the memory cycles consumed by on-chip communication significantly decreases without substantially increasing off-chip misses. For instance, by devoting 10% of capacity to replicas, the total L1 miss-cycles-per-instruction decreases by at least 0.4 cycles. Further increasing the replica capacity beyond 10% has varied results. Increasing the percent capacity for replicas to 30% reduces Oltp's total L1 miss-cycles-per-instruction by an additional 0.2 cycles, while Jbb and Zeus observe less than a 0.1 additional reduction. In contrast, due to replication conflicting with Apache's large working set (Figure 4-5), increasing the percent capacity for replicas beyond 10% increases Apache's total L1 miss-cycles-per-instruction by as much as 1.3 cycles.

For the scientific workloads, replication has little performance benefit and instead can lead to substantial performance degradation. For example, Apsi and Ocean have little shared read-only data activity. Thus, replication has no effect on performance causing these workloads to be indistinguishable from the y-axis on Figure 4-14 or Figure 4-15. On the other hand, Barnes has some highly localized shared read-only data activity. Thus, increasing replication initially reduces total L1 miss-cycles-per-instruction by 0.2 cycles and then further replication increase has no performance impact. Finally, increasing Art's replica capacity

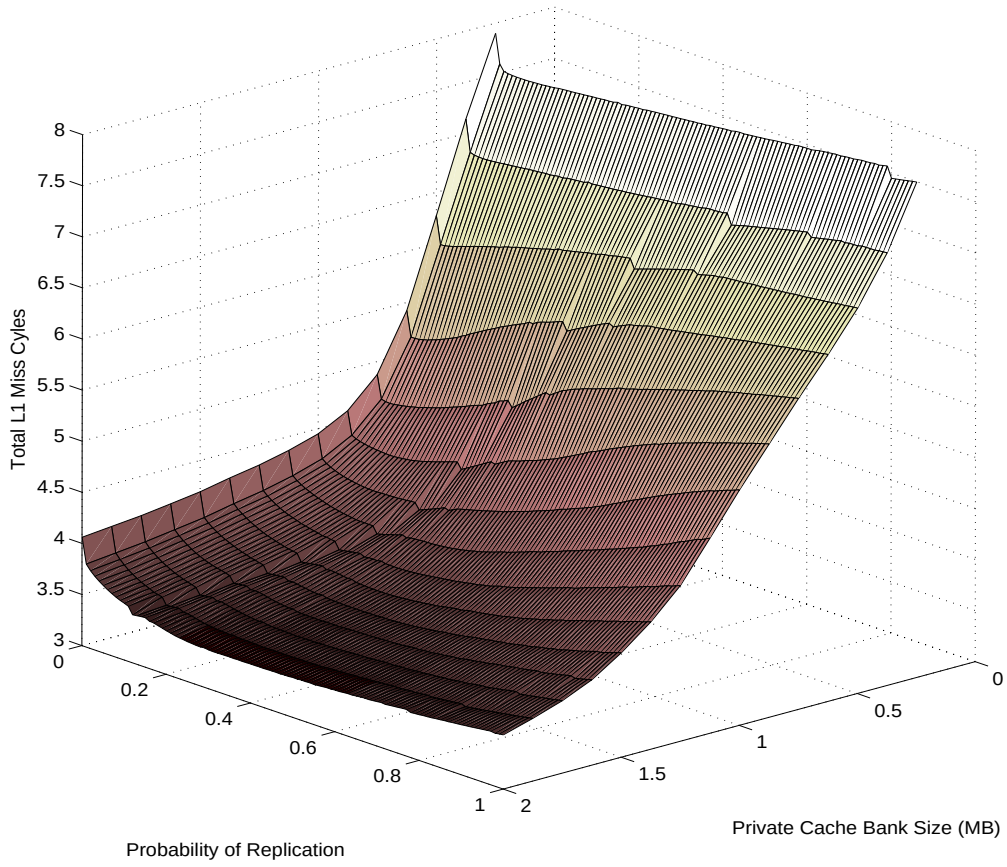


FIGURE 4-16. Replication Model: *Apache* Cache Capacity vs. Probability of Replication

beyond 4% leads to a 17 cycle performance degradation. This large performance degradation is caused by replication evicting Art's critical 8 MB working set (Figure 4-5). Art's abrupt change in performance highlights replication's impact on effective cache capacity.

The optimal amount of replication shifts depending on the relationship between working set size and L2 cache capacity. Figure 4-16, Figure 4-17, and Figure 4-18 provide 3D surface plots of how replication and cache size interact to affect performance. The amount of replication is shown as the probability of replication (Equation 4.19), instead of the percent capacity for replicas, in-order to directly compare various cache sizes with different relative amounts of replicated data. For small (< 1 MB) private caches, Apache's

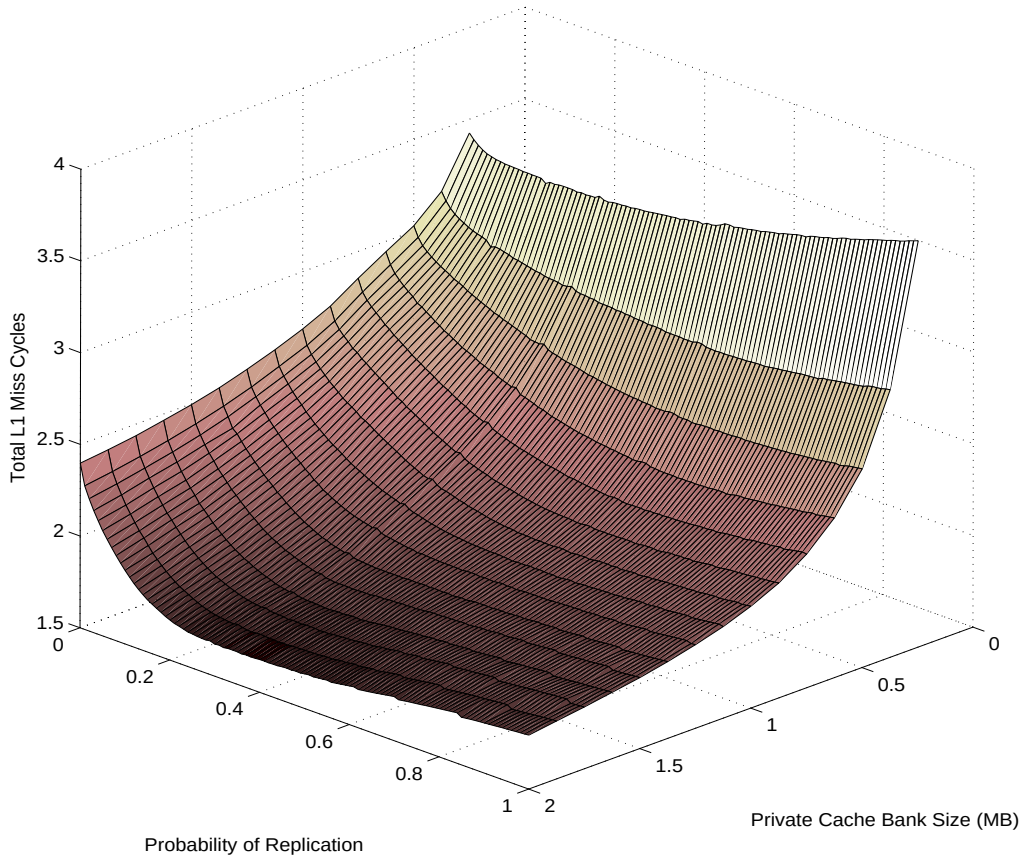
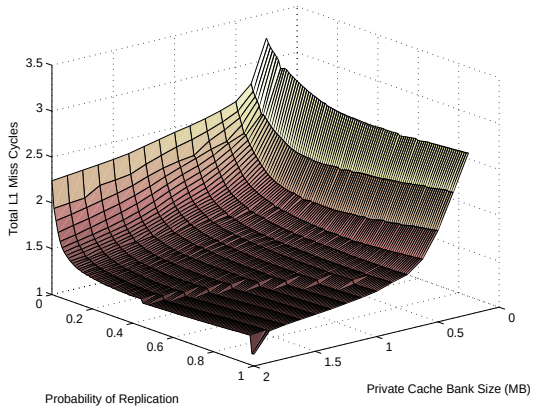


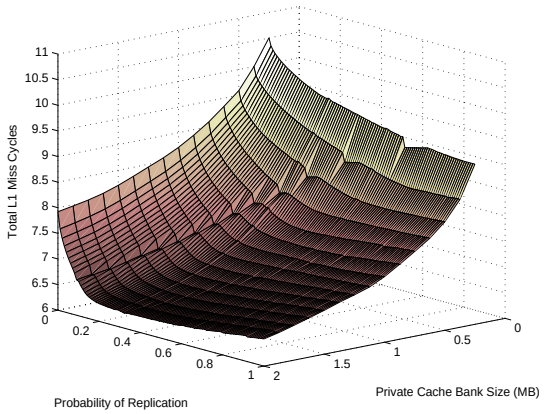
FIGURE 4-17. Replication Model: *OLTP* Cache Capacity vs. Probability of Replication

optimal probability of replication is less than 0.1 (Figure 4-16). Then as the cache size increases, the benefit provided by replication increases, and Apache's optimal probability of replication nears 0.5.

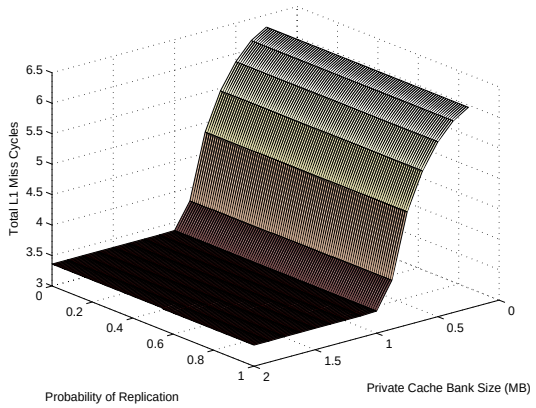
Oltp (Figure 4-17) demonstrates more diverse behavior than Apache. For very small (< 0.25 MB) private caches, Oltp's optimal probability of replication is near 0.1. Then, for private caches between 0.5-1 MB, Oltp prefers replication between 0.4-0.6 in order to move its critical shared read-only working set close to multiple processors. Finally, for private caches between 1-2 MB, Oltp prefers replication between 0.2-0.4 because it requires less replication to keep its critical shared read-only working set close and therefore can utilize more capacity for unique data.



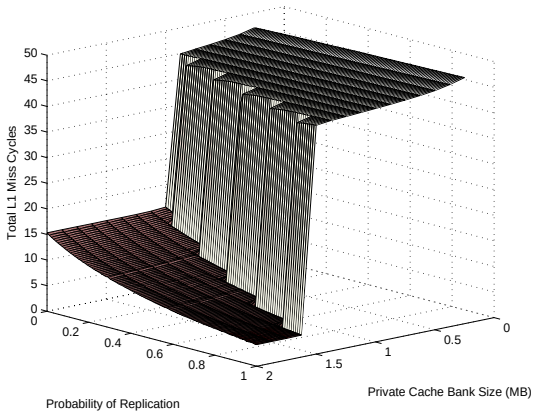
Jbb



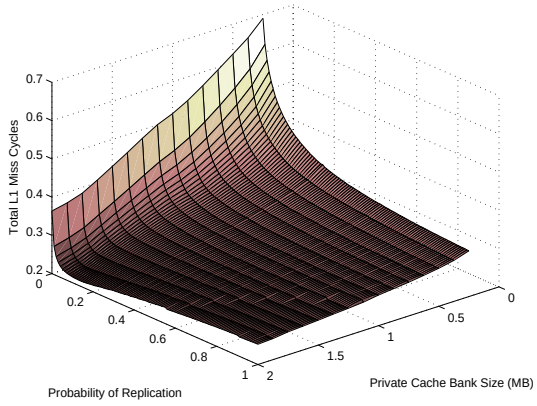
Zeus



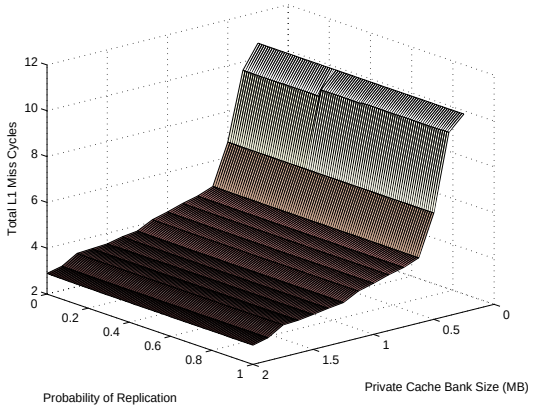
Apsi



Art



Barnes



Ocean

FIGURE 4-18. Replication Model: Cache Capacity vs. Probability of Replication

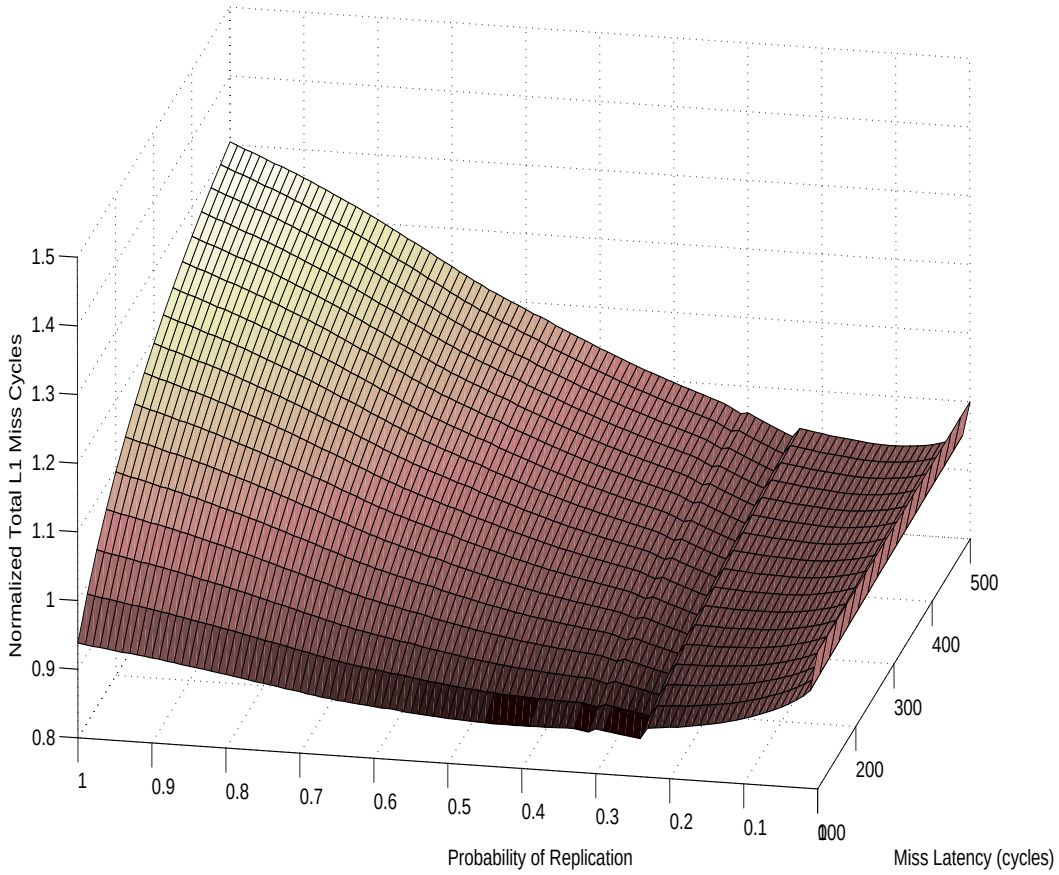


FIGURE 4-19. Replication Model: *Apache* Miss Latency vs. Probability of Replication

Figure 4-18 presents the remaining six workloads' surface plots. The four surface plots or Jbb, Zeus, Art and Barnes exhibit dynamic shape and the optimal probability of replication shifts depending on cache size. Conversely, Apsi and Ocean show no change in their optimal probability of replication.

The optimal amount of replication also shifts depending on the relationship between workload behavior and miss latency. Figure 4-19, Figure 4-20, and Figure 4-21 illustrate that the optimal amount of replication increases as miss latency decreases. In order to observe the relative performance trends, the plots normalize total L1 miss-cycles-per-instruction to the zero replication case for each separate miss latency value. Also both plots utilize an 8 MB aggregate L2 cache. For Apache (Figure 4-19) with the long 500 cycle miss latency, maximum replication encounters a 30% performance degradation versus no replication,

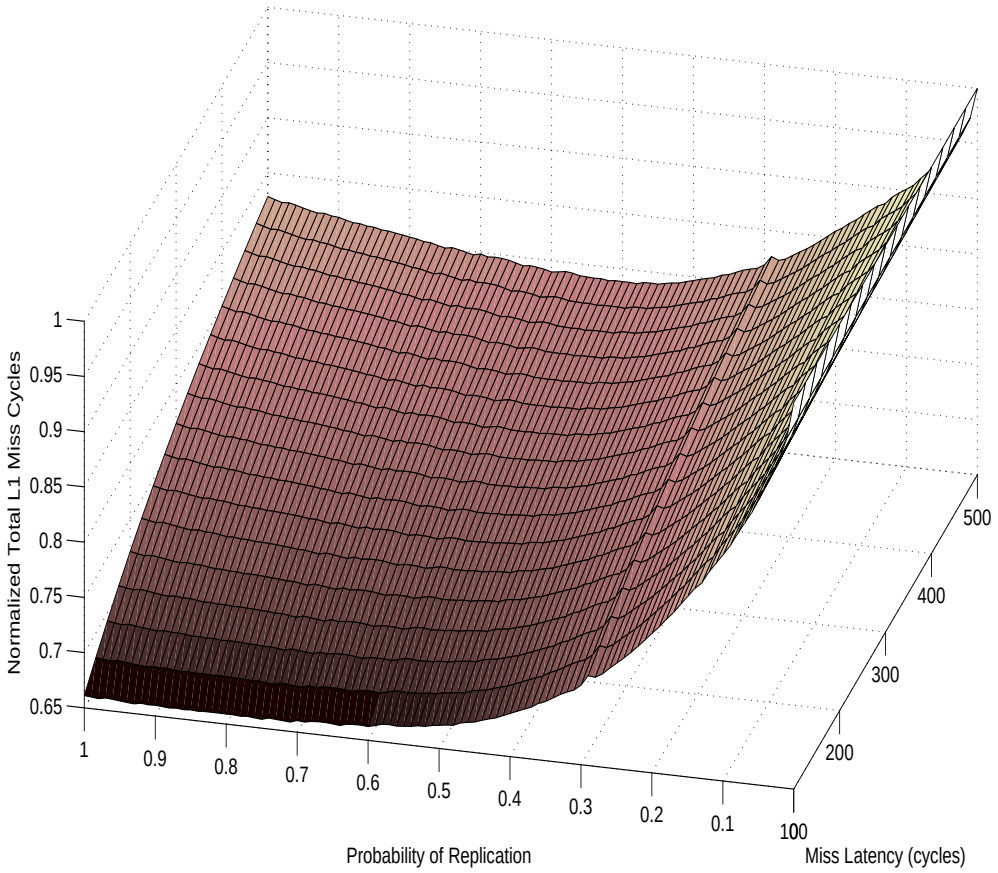
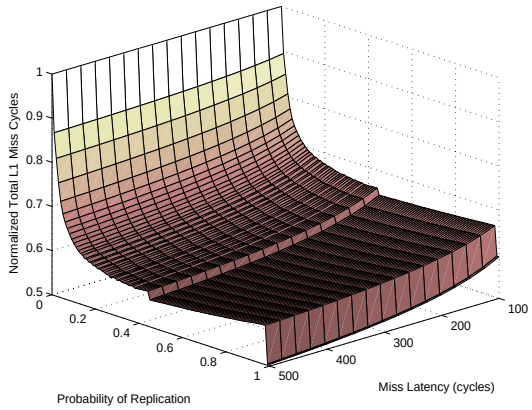
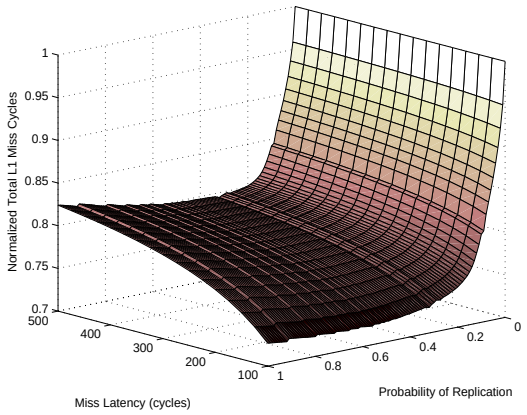


FIGURE 4-20. Replication Model: *OLTP* Miss Latency vs. Probability of Replication

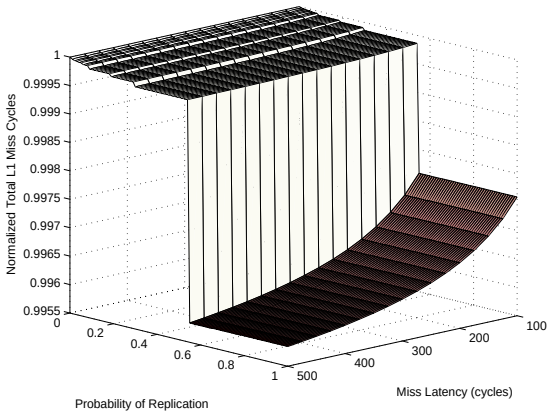
while at 100 cycles maximum replication achieves a 7% speedup. Furthermore, Apache's lowest point shifts from 0.1 probability of replication for the 500 cycle miss latency to 0.25-0.45 probability of replication for the 100 cycle miss latency. Similarly, Oltp's (Figure 4-20) lowest point shifts from 0.5 probability of replication for the 500 cycle miss latency to maximum replication for the 100 cycle miss latency. Finally, Figure 4-21 presents the remaining six workload's surface plots. In particular, Zeus, and Barnes demonstrate similar dynamic shape, with the optimal probability of replication shifting to smaller values as latency increases. Meanwhile, Jbb, Apsi, Art, and Ocean show little sensitivity to miss latency and always prefer maximum replication.



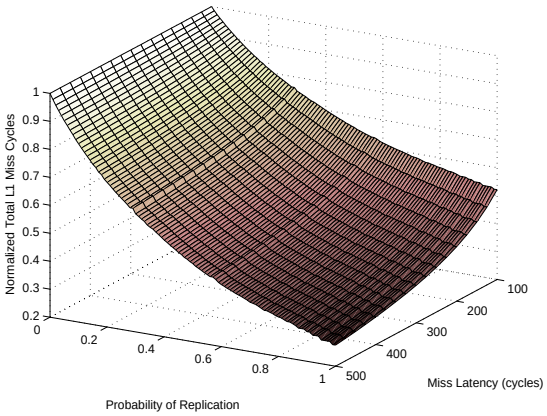
Jbb



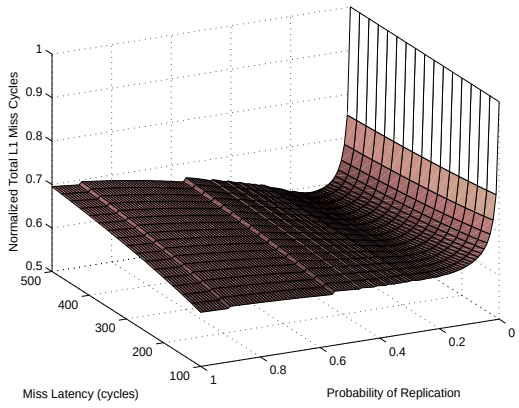
Zeus



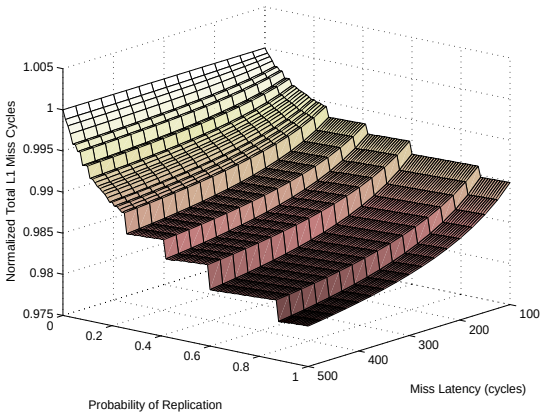
Apsi



Art



Barnes



Ocean

FIGURE 4-21. Replication Model: Miss Latency vs. Probability of Replication

4.4 Summary

This chapter demonstrated the potential of exploiting workload behavior to reduce on-chip wire delay. The chapter first characterized commercial and scientific workload behavior, and then applied the collected characterization data to two abstract analytical models. The models are not detailed enough to accurately predict absolute performance, but do provide high-level intuition of migration's and replication's performance impact. For instance, the first model showed migration can improve scientific workload performance, but inter-processor sharing limits migration's benefit in commercial workloads. The second replication model revealed that selectively replicating frequently requested blocks provides better performance than naively replicating all shared blocks, and that the optimal amount of replication varies between workloads. Furthermore, the replication model illustrated that cache capacity and memory latency significantly impact the optimal replication amount. Thus, an adaptive selective replication policy that can identify and adjust to changes in replication's benefit, can provide more robust performance.

Chapter 5

Cache Block Migration

Using an analytical model, the previous chapter estimated migration may substantially improve workload performance where little data sharing exists, but may degrade workload performance where significant sharing exists. Using full system simulation, this chapter confirms that indeed sharing limits migration's performance benefit. Also this chapter demonstrates other aspects, such as smart searches and off-chip misses, further limit migration's performance benefit. The chapter begins with Section 5.1 motivating dynamic migration and Section 5.2 describing the shared cache design. Then Section 5.3 explains the migrating cache design, Section 5.4 details the evaluation methodology, and Section 5.5 provides simulation results. Finally, Section 5.6 concludes the chapter.

5.1 Motivation

Increasing wire delay makes it difficult to provide uniform access latencies to all on-chip L2 cache banks. One alternative is Kim *et al.'s* Non-Uniform Cache Architecture (NUCA) design [58], which allow nearer cache banks to have lower access latencies than further banks. For a uniprocessor, Kim *et al.* demonstrated that dynamically migrating frequently requested blocks to these nearer cache banks significantly improved performance by effectively exploiting the distance locality between the processor and close cache banks.

However, supporting multiple processors (e.g., 8) places additional demands on NUCA cache designs. First, simple geometry dictates that eight regular-shaped processors must be physically distributed across the 2-dimensional die. A cache bank that is physically close to one processor cannot be physically close to

TABLE 5-1. 2010 System Parameters

Memory System		Dynamically Scheduled Processor	
split L1 I & D caches	64 KB, 4-way, 3 cycles	clock frequency	5.0 GHz
unified L2 cache	16 MB, 16-way pseudoLRU[95]	reorder buffer / scheduler	128 / 64 entries
L1/L2 cache block size	64 Bytes	pipeline width	4-wide fetch & issue
memory latency	250 cycles + on-chip delay	pipeline stages	15
memory bandwidth	50 GB/s	direct branch predictor	3.5 KB YAGS
memory size	4 GB of DRAM	return address stack	64 entries
outstanding memory requests/CPU	16	indirect branch predictor	256 entries (cascaded)

all the others. Second, an 8-way CMP requires eight times the sustained cache bandwidth. These two factors strongly suggest a physically distributed, multi-port NUCA cache design.

While block migration works uniformly well for uniprocessors, this chapter confirms Chapter 4's conclusion that shared CMP cache migration performance depends on inter-processor sharing. Additionally, the chapter exposes that, because processors pull frequently requested cache blocks in multiple directions, migration's effectiveness in a shared CMP cache is more dependent on "smart searches" [58] than its uniprocessor counterpart, yet smart searches are harder to implement in a CMP environment. Specifically, this chapter shows that CMP migration without a smart search mechanism degrades performance versus the baseline CMP cache for all workloads except Barnes. Furthermore, even with an perfect search mechanism, block migration alone only improves the performance of four workloads—Jbb, Art, Barnes, and Ocean—by a maximum of 2%, while degrading the performance of the other four workloads by as much as 7%. This is in part because shared blocks migrate to the middle equally-distant cache banks, accounting for 55-83% of L2 hits for the commercial workloads.

5.2 Baseline: CMP-SNUCA

The chapter targets eight-processor CMPs assuming the 45 nm technology generation projected in 2010 [37]. Table 5-1 specifies the system parameters for all designs. Each CMP design assumes approximately

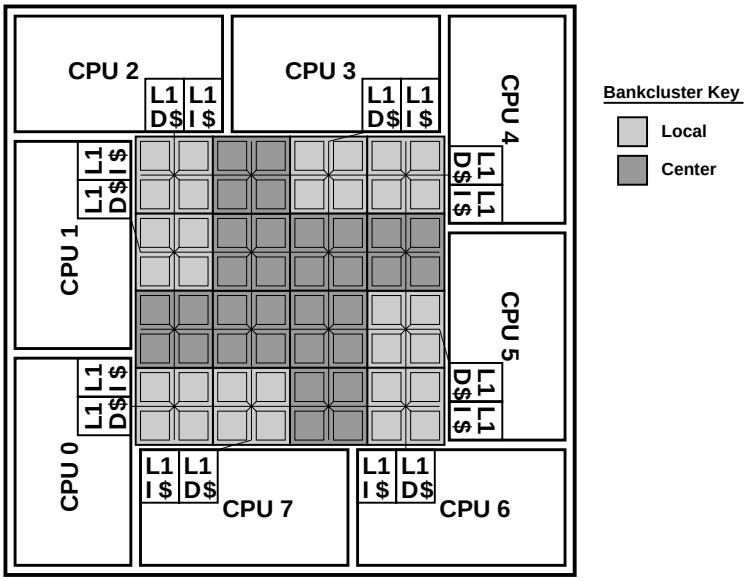


FIGURE 5-1. 16 MB CMP-NUCA Layout with CMP-DNUCA Bankcluster Regions

300 mm^2 of available die area [37]. We estimate eight 4-wide superscalar processors would occupy 120 mm^2 [64] and 16 MB of L2 cache storage would occupy 64 mm^2 [37]. The on-chip interconnection network and other miscellaneous structures occupy the remaining area.

Figure 5-1 illustrates the baseline design derived from Kim, *et al.'s* S-NUCA-2 design [58]. The baseline is denoted CMP-SNUCA because it utilizes static block placement. The CMP-NUCA layout differs from the original uniprocessor NUCA design in several important ways:

- To correspond to the data presented in Section 3.2.1 of Chapter 3, the 16 MB L2 cache is partitioned into 64 banks to control bank access latency [1] and to provide sufficient bandwidth to support. This configuration differs from the original NUCA [58] and CMP-NUCA [15] proposals, which assumed an aggressive 8 FO4 cycle time [47] and thus splitted the cache into 256 banks to provide the optimal balance between intra-bank and inter-bank latency. Due to recent studies advocating that frequency scaling is slowing down due to power constraints [102], We assume a 20 FO3 delay. The slower frequency allows for less pipelining and partitioning in the NUCA network.

- Second, the evaluated CMP-NUCA implementation connects four banks to each switch and expands the link width to 64 bytes. Each 2.2 mm link has a latency of 2 cycles and each switch has a 1 cycle latency. The wider CMP-NUCA network provides the additional bandwidth needed by an 8-processor CMP.
- Third, all CMP cache banks utilize 16-way set-associative banks with a pseudo-LRU replacement policy [95] to reduce contention from different processors' working sets [70].
- Finally, we assume an idealized off-chip communication controller to provide consistent off-chip latency for all processors.

5.3 CMP-DNUCA

5.3.1 Overview

CMP-DNUCA migrates frequently accessed blocks to reduce cache access latency. Section 4.1.1 illustrated a majority of requests are for a small percentage of blocks. CMP-DNUCA strives to move these most frequently requested blocks towards the requesting processor. Specifically, this dissertation's CMP-DNUCA implementation employs block migration within the previously described baseline CMP-NUCA layout. Three essential design aspects of a cache employing block migration are: 1. the policy for initially *allocating* blocks into the cache, 2. the policy for *migrating* blocks within the cache, and 3. the policy for *searching* the cache. CMP-DNUCA strives to reduce additional state, while providing correct and efficient allocation, migration, and search policies.

5.3.2 Implementation

Allocation. CMP-DNUCA permits block migration by utilizing logical and physical organizations similar to the uniprocessor D-NUCA design [58]. CMP-DNUCA permits block movement by *logically* sepa-

rating the L2 cache banks into 4 unique banksets, where an address maps to a bankset and can reside within any bank of the bankset. CMP-DNUCA *physically* separates the cache banks into 16 different *bankclusters*, each containing one bank from every bankset in a two-by-two array. The bankclusters are grouped into two equal regions: local—lightly shaded in Figure 5-1—and center—darker shaded in Figure 5-1. The unique local bankcluster closest to each processor is identified the processor's *my local* bankcluster. Similarly, the unique center bankcluster closets to each processor is defined as the processor's *my center* bankcluster. From the perspective of each processor, the remaining bankclusters are identified as *other local* and *other center* bankclusters. Ideally, block migration would maximize L2 hits within each processor's my local bankcluster where the uncontended L2 hit latency (i.e., load-to-use latency) is the lowest and limit hits to other local and center bankclusters, where the uncontended latency is significantly higher.

The allocation policy seeks an efficient initial placement for a cache block, without creating excessive cache conflicts. While 16-way set-associative banks help reduce conflicts, the interaction between migrations, replications, and allocations can still cause unnecessary replacements. CMP-DNUCA implements a simple, static allocation policy that uses the low-order bits of the cache tag to select a bank within the block's bankset (i.e., the bankcluster). This simple scheme is non-optimal because allocations are evenly distributed across all bankclusters despite the fact that some bankclusters are more utilized than others. However, bankcluster utilization varies depending on the workload and movement policy, so the wide distribution of allocations often avoids the pathological case where the majority of allocations are to the same bankcluster storing the most valuable data. While not studied in this document, we conjecture that static allocation also works well for heterogeneous workloads, because all active processors will utilize the entire L2 cache storage.

Movement. We investigated several different movement policies for CMP-DNUCA. A movement policy should maximize the proportion of L2 hits satisfied by the banks closest to a processor. Section 4.2's analytical model assumed a simple policy where blocks directly migrated to a requesting processor's local

bankcluster. This direct migration policy increases the number of local bankcluster hits, if most cache blocks are private to a particular processor. However, if blocks are shared, direct migration also increases the proportion of costly remote hits satisfied by other local bankclusters. Instead CMP-DNUCA implements a simple gradual migration policy that moves blocks along the four bankcluster chain:

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

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

Search. The best performing uniprocessor D-NUCA search policy used a two-phase multicast search. CMP-DNUCA uses a similar two-phase search policy that strives to maximize first phase hits while limiting the number of messages. Under the gradual migration policy, hits most likely occur in one of nine bankclusters: the requesting processor's local bankcluster, or the eight center bankclusters. Therefore the first phase of our search policy broadcasts a request to the appropriate banks within these nine bankclusters. If all nine initial bank requests miss, the request is sent to the remaining seven banks of the bankset. Only after a request misses in all 16 banks of the bankset will a request be sent off chip. Waiting for 16 replies over two phases adds significant latency to cache misses.

To reduce the latency of detecting a cache miss, the uniprocessor D-NUCA design utilized a "smart search" [58] mechanism using a partial tag array. The centrally-located partial tag structure [56] replicated the low-order bits of each bank's cache tags. If a request missed in the partial tag structure, the block was

guaranteed not to be in the cache, i.e. false negatives equaled zero. This smart search mechanism allowed nearly all cache misses to be detected without searching the entire bankset.

In CMP-DNUCA, adopting a partial tag structure appears impractical. All processors cannot quickly access a centralized partial tag structure due to wire delays. Fully replicated 6-bit partial tag structures (as used in uniprocessor D-NUCA [58]) require 1.5 MB of state, an extremely high overhead. More importantly, separate partial tag structures require a complex coherence scheme that updates address location state in the partial tags with block migrations. However, because architects may invent a solution to this problem, we evaluate CMP-DNUCA both with and without a perfect search mechanism in Section 5.5.

A unique problem of CMP-DNUCA is the potential for *false misses*, where L2 requests fail to find a cache block because it is in transit from one bank to another. Because CMP-DNUCA uses token coherence [74, 76], false misses don't create a correctness problem. False misses, like other data races in token coherence, will activate the persistent request mechanism. However, persistent request activation is slow and frequent false misses could be a performance problem.

We significantly reduced the frequency of false misses by implementing a lazy migration mechanism. We observed that almost all false misses occur for a few hot blocks that are rapidly accessed by multiple processors. By delaying block migrations by a thousand cycles, and canceling migrations when a different processor accesses the same block, CMP-DNUCA still performs at least 75% of all scheduled migrations with only a 2-14% increase in persistent requests.

5.4 Methodology

Both the CMP-SNUCA and CMP-DNUCA designs implement a CMP-Token cache-coherence protocol [76] with 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 allows the L2 cache to have no transient states. The inter-chip coherence protocol maintains directory state at the off-chip memory controllers and only tracks which CMP nodes contain valid block copies. To facilitate fast cache-to-cache transfers of shared read-write data, the protocol implements migratory sharing [99]. To deal with races that exist in a CMP-Token protocol, all designs utilize a distributed persistent request mechanism [76] that activates after a 400-cycle timeout latency. 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 [105], except it issues prefetches for both stores, as well as loads, due to the stronger memory consistency model.

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 5-1. Network routing is performed using a virtual cut-through scheme with finite buffering at the switches. Most buffers are constrained to three messages. The buffers between the on-chip and off-chip networks are infinitely sized to decouple the on-chip network from off-chip queueing delay due to limited off-chip bandwidth.

The workloads evaluated in this chapter are the same as those in Chapter 4. However, to account for the non-determinism that exists in multi-threaded workloads [3], all simulations contain small random perturbations in the memory latency and the error bars indicate the 95% confidence interval.

5.5 Evaluation

Dynamic block migration strives to reduce the NUCA cache hit latency by moving frequently-accessed blocks. However, migration's unbalanced cache bank utilization increases the off-chip miss rate, leading to a nominal overall performance benefit even with a perfect smart search mechanism. First, Section 5.5.1 illustrates dynamic block migration's success in moving blocks to the quickly accessible cache banks.

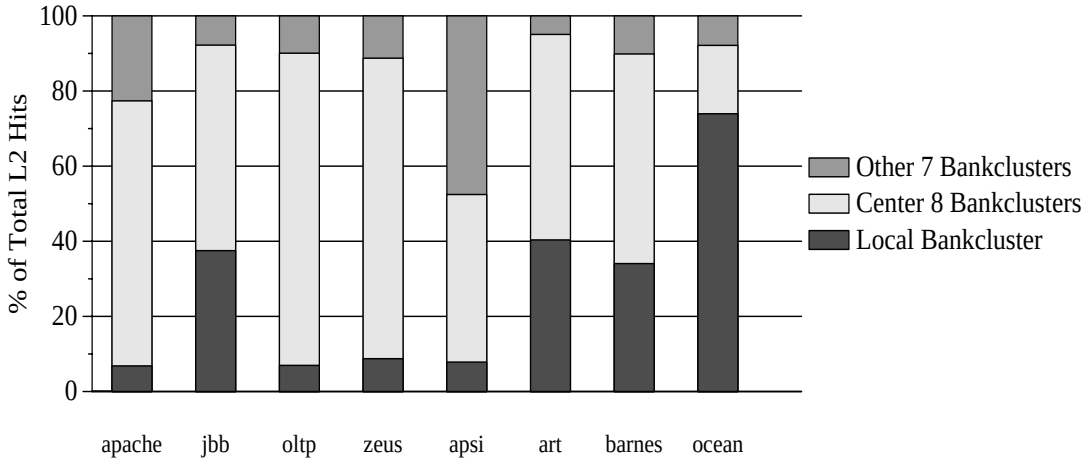


FIGURE 5-2. CMP-DNUCA: L2 Hit Distribution

Then Section 5.5.2 demonstrates CMP-DNUCA relies upon a smart search mechanism to achieve lower L2 hit latency and to avoid inflating the miss penalty.

5.5.1 Block Movement in CMP-DNUCA

CMP-DNUCA's migration policy significantly affects the hit clustering within the CMP-NUCA cache. In particular, the high degree of sharing in the commercial workloads restricts block migration's benefit. Figure 5-2 shows the center bankclusters satisfy 55-83% of L2 hits for the four commercial workloads. The high number of central hits directly relates to the increased sharing in the commercial workloads—Section 4.1.3. Figure 5-3a graphically illustrates CMP-DNUCA's L2 hit distribution for Oltp. The top plot of Figure 5-3a displays the L2 hit distribution for all processors. The dark grey squares in the top plot indicate the majority of hits are satisfied by blocks that congregate in the center. Specifically for Oltp, the 8 center bankclusters satisfy over 83% of all L2 hits. The lower 8 plots of Figure 5-3a separate the hit distribution on a per processor basis. The light grey squares on the periphery of the cache indicate each processor's local bankcluster satisfies a more than requests than the seven remote bankclusters. Specifically, each processor's local bankcluster satisfies 8% of Oltp misses, while the seven remote bankclusters satisfy 10% of Oltp misses.

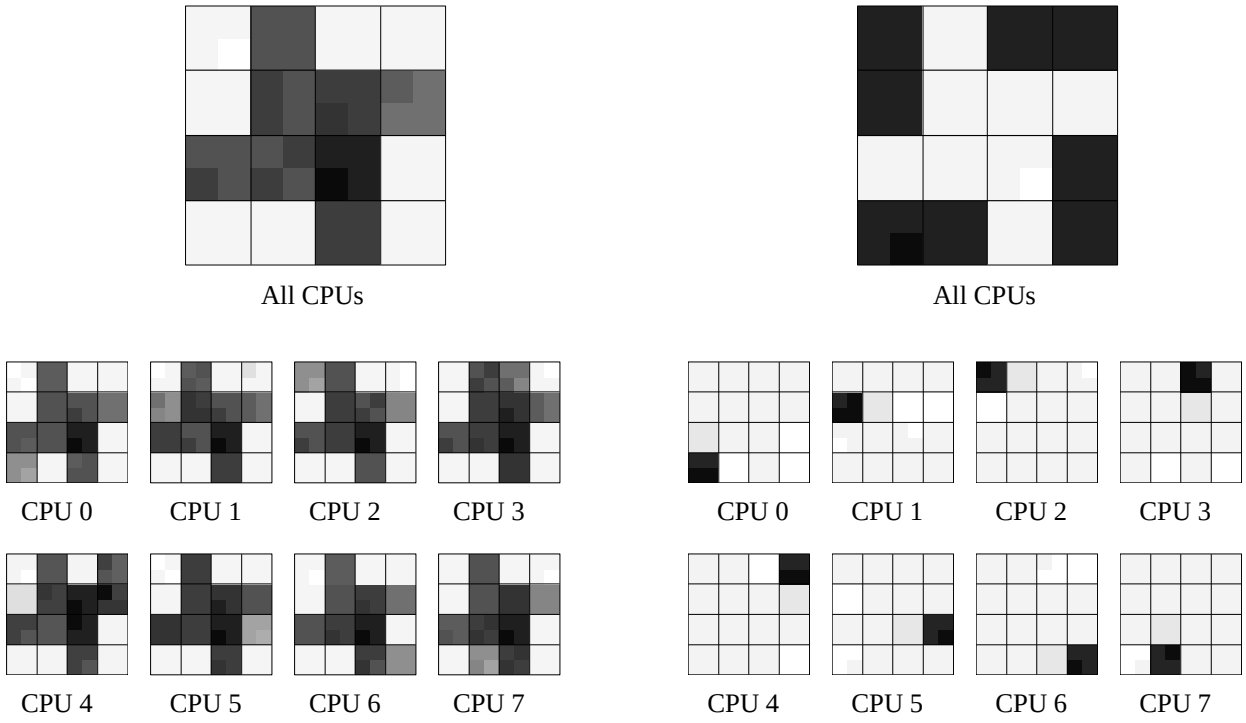


FIGURE 5-3. a) Oltp DNUCA Distribution **FIGURE 5-3. b) Ocean DNUCA Distribution**

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

Conversely, CMP-DNUCA designs exhibit very different behavior for the four scientific workloads. The scientific workload Ocean iterates over a column-blocked 2D matrix resulting in good locality and little sharing. Figure 5-2 indicates that CMP-DNUCA successfully migrates 74% of Ocean's L2 hits to the local bankclusters. The dark colored squares of Figure 5-3b graphically display how well CMP-DNUCA is able to split Ocean's data set into the local bankclusters. To a lesser degree, CMP-DNUCA is also able to split Art's and Barnes's working set, with the local bankclusters satisfying 40% and 34% of L2 hits, respectively. Meanwhile, Apsi experiences virtually no improvement from CMP-DNUCA, which starkly contradicts migration's projected improvement modeled in Chapter 4. The main reason for this disparity is 88% of L2 blocks allocate in Apsi are requested only twice. Therefore, CMP-DNUCA's distributed allocation policy and gradual migration policy don't move blocks to the requesting processor's local bank during

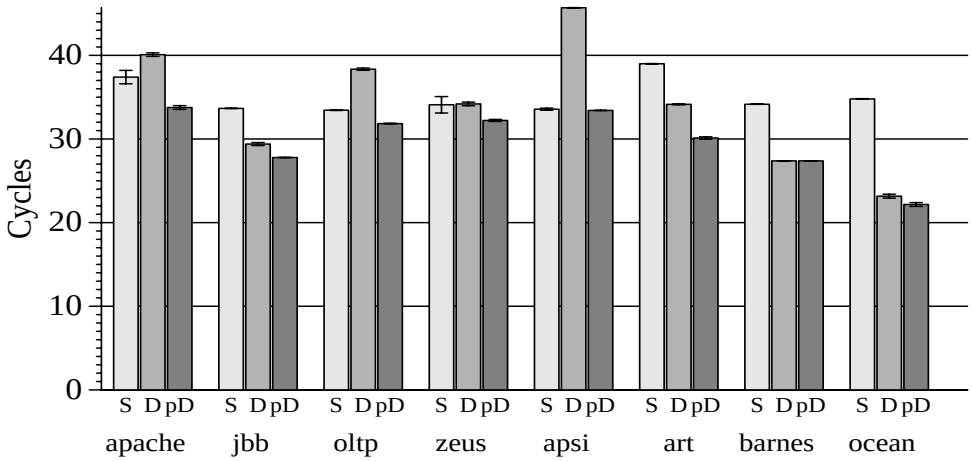


FIGURE 5-4. CMP-DNUCA: Average L2 Hit Latency
(S: CMP-SNUCA, D: CMP-DNUCA, pD: perfect CMP-DNUCA)

their brief active lifetimes. Instead, Chapter 4's private cache allocation policy and direct migration policy better matches Apsi's request behavior.

5.5.2 Searching in CMP-DNUCA

While block migration has the potential to reduce hit latency by moving blocks to the closer cache banks, the slow two-phase search policy actually causes L2 hit latency to increase for some workloads. Figure 5-4 shows that CMP-DNUCA reduces L2 hit latency versus CMP-SNUCA for the four workloads, Jbb, Art, Barnes, and Ocean. The degree of latency reduction directly relates to the success DNUCA has moving L2 hits to the local bankclusters. For instance, Figure 5-2 shows DNUCA greatest success exists in Ocean, which also is the workload that encounters the most significant L2 latency reduction in Figure 5-4. For the other four workloads: Apache, Oltp, Zeus, and Apsi, the latency increase results from a combination of a much lower percentage of local bankcluster hits and a higher percentage of second phase hits in the remote bankclusters. Specifically, second phase hits encounter 31 to 51 more delay cycles than CMP-SNUCA L2 hits because of the delay waiting for miss responses from the first phase requests.

A smart search mechanism would solve this problem. Figure 5-4 shows the L2 hit latency attained by CMP-DNUCA with perfect search (perfect CMP-DNUCA), where a processor sends a request directly to the cache bank storing the block. Perfect CMP-DNUCA reduces L2 hit latency by 0-13 cycles versus CMP-SNUCA. Furthermore, when the block isn't on chip, perfect CMP-DNUCA immediately generates an off-chip request, allowing its L2 miss latency to match that of CMP-SNUCA. Although the perfect search mechanism is infeasible, architects may develop practical smart search schemes in the future [92]. The rest of this section evaluates CMP-DNUCA with and without perfect searches to examine the potential benefits of dynamic block migration in CMP-NUCA.

With perfect searches, CMP-DNUCA achieves lower L2 hit latencies than CMP-SNUCA, however, perfect CMP-DNUCA encounters more off-chip misses than CMP-SNUCA. Figure 5-5 presents the L1 miss latency to on-chip single requestor, shared read-only, and shared read-write blocks. As expected, migration significantly reduces single requestor latency—up to 46% for perfect CMP-DNUCA—, but is less effect reducing shared block latency—maximum 23%. Interestingly, migration decreases shared read-write latency by 5-23%, while migration has virtually no effect on shared read-only latency. Corresponding to Chapter 4's workload characterization, the reason for the disparity is shared read-only blocks are more finely shared between processors than shared read-write blocks. Therefore, shared read-only blocks are more likely to congregate in the center than shared read-write blocks. Figure 5-5 also displays migration increases off-chip misses. In particular, by moving the most frequently requested blocks to a subset of banks (Figure 5-2), CMP-DNUCA's static allocation policy is more likely to displace active cache blocks than CMP-SNUCA. Specifically, all workloads except Apsi encounter 4-21% more off-chip misses for CMP-DNUCA than CMP-SNUCA. This behavior directly corresponds to Figure 5-2 where, for all workloads except Apsi, certain bankclusters satisfied a larger than average percentage of L2 hits.

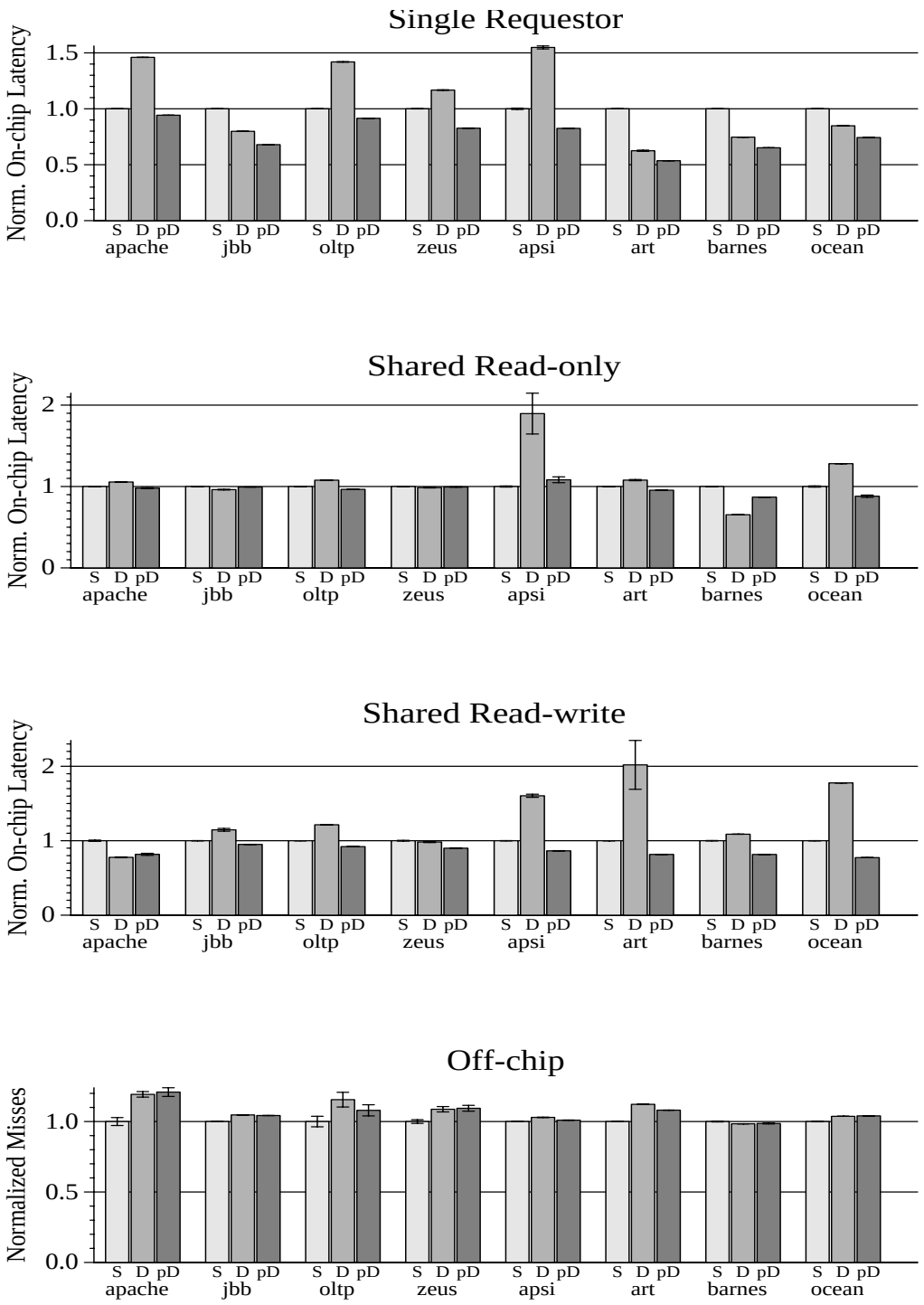


FIGURE 5-5. Normalized L1 Miss Latency to Sharing Types and Off-chip Misses (S: CMP-SNUCA, D: CMP-DNUCA, pD: perfect CMP-DNUCA)

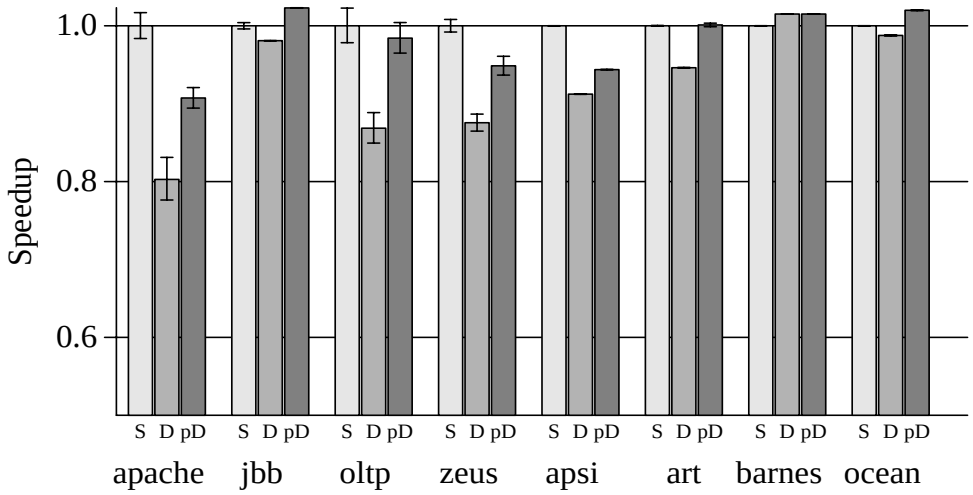


FIGURE 5-6. CMP-DNUCA: Speedup
(S: CMP-SNUCA, D: CMP-DNUCA, pD: perfect CMP-DNUCA)

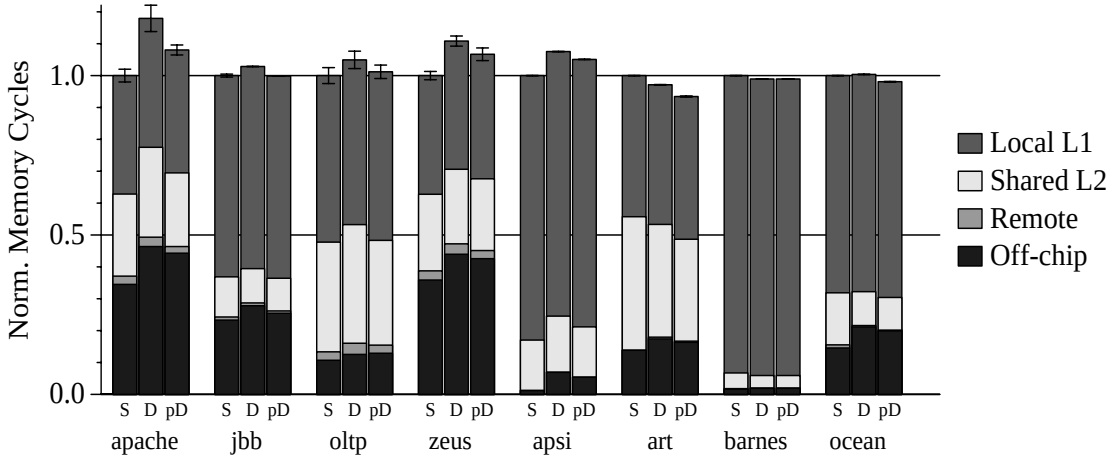


FIGURE 5-7. CMP-DNUCA: Normalized Memory Cycles
(S: CMP-SNUCA, D: CMP-DNUCA, pD: perfect CMP-DNUCA)

CMP-DNUCA's off-chip miss increase cancels its reduction in L2 hit latency, leading to little overall performance improvement. Figure 5-6 displays the speedup of CMP-DNUCA and perfect CMP-DNUCA over that of CMP-SNUCA. The maximum speedup achieved by CMP-DNUCA is 1% for Barnes and the maximum speedup by perfect CMP-DNUCA is 2% for the workloads Jbb and Ocean. For most other workloads, both CMP-DNUCA and perfect CMP-DNUCA experience a slowdown versus CMP-SNUCA. Apache encounters the largest performance degradations, with CMP-DNUCA degrading performance by 18% and perfect CMP-DNUCA degrading performance by 7%. While perfect CMP-DNUCA reduced L2

hit latency by 4 cycles in Apache, the reduction was not enough to compensate for the 16% increase in off-chip misses. Figure 5-7 summarizes the benefits and cost migration in the CMP-NUCA cache by breaking down the cycles spent in the memory hierarchy. The ‘L1 Cache’ bars display the fraction of the average memory access time contributed by L1 cache hits. The ‘Shared L2’ category presents the cycles spent on L2 hits. The ‘Remote’ category represents the cycles spent on requests that hit in remote L1 caches. Finally, the ‘Memory’ bar exposes the cycles spent on memory accesses.

Overall, prefect CMP-DNUCA reduces the cycles spent on L2 hits at the cost of increasing the cycles spent on memory requests. To achieve substantial performance improvement over the baseline, L2 hit latency must be reduced without significantly affecting the amount of off-chip misses. For instance, Section 4.1.4 revealed the majority of read-only requests where to a small fraction of read-only blocks. By carefully replicating a small number of L2 blocks, one could obtain most of replication’s benefit without significantly increasing off-chip misses (Chapter 6). Another solution is to replace slow conventional wires with fast on-chip transmission lines (Chapter 7).

5.6 Summary

This chapter demonstrated that inter-processor sharing limits the performance benefit of dynamically migrating cache blocks within a shared CMP cache. Additionally, the chapter showed that CMP cache migration is more dependent on “smart searches” [58] than the original uniprocessor proposal, and that migration can increase off-chip misses, thus cancelling its on-chip latency benefit. In general, migration’s meager performance benefit does not justify its implementation complexity. Instead, a private CMP cache can statically incorporate migration’s latency benefit without CMP-DNUCA’s complexity. Furthermore, through replication, a private cache can decrease hit latency to shared L2 blocks. However, for private caches to be overall beneficial, one must ensure replication’s reduction in effective cache capacity does not cancel its benefit.

Chapter 6

Adaptive Selective Replication

By allowing multiple L2 cache block copies, replication can minimize the cache access time encountered by separate on-chip processors. However, too much replication can significantly reduce effective cache capacity, thus increasing off-chip misses. To attack these conflicting trends, this chapter describes *Adaptive Selective Replication (ASR)*, a mechanism that dynamically monitors workload behavior to control replicating cache blocks within a private CMP cache hierarchy.

This chapter begins with Section 6.1 motivating the need to adapt L2 cache replication to workload behavior. Next, Section 6.2 introduces the baseline private CMP cache design that allows replication between different local L2 caches. Then Section 6.3 describes the ASR algorithm and Section 6.4 presents the ASR implementation. Section 6.5 follows with details of the evaluation methodology and Section 6.6 provides simulation results. Finally, Section 6.7 discusses related work and Section 6.8 concludes.

6.1 Motivation

L2 cache block replication presents both a key opportunity and a significant challenge to provide good CMP performance for a wide variety of workloads. In particular, replication can reduce cache access latency by copying data close to multiple processors, but can increase off-chip misses by decreasing the effective cache capacity. Currently CMP systems either employ shared L2 caches [32, 62] that prevent replication, or private L2 caches [65, 77] that allow all shared data to be replicated.

Recent hybrid cache proposals seek to achieve a balance between latency and capacity by selectively replicating cache blocks. Cooperative Caching [20], CMP-NuRapid [26], and Victim Replication [121] 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, CMP-NuRapid and Victim Replication each have a single static replication policy that cannot dynamically adapt to different workload and data set behavior. Cooperative Caching uses a configurable probability to tradeoff replication with effective cache capacity, but does not propose a method to adjust the probability.

Previously, Chapter 4 advocated the need for an adaptive replication policy. Specifically, Section 4.3 showed through its analytical model that different workloads preferred different amounts of replication. Furthermore, the 3D plots of Section 4.3.2 illustrated that for some workloads the optimal replication level changes depending on the interaction between workload behavior and system constraints. Clearly, some adaptive policy is needed to determine the best replication level for given combinations of workloads and systems.

This chapter describes 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.

6.2 Baseline: Private CMP Caches

This chapter evaluates replication within a private CMP cache composed of eight banks as illustrated in Figure 6-1. Similar to the 8-processor Sun Niagara CMP [62], the private CMP layout groups the 8 on-chip processors into two equal groups and places each processor group on opposite edges of the chip. Though the private cache uses larger and slower cache banks than Chapter 5's shared NUCA cache, the coarse-grained cache has several advantages. For instance, splitting the private cache into 8 banks, instead of 64,

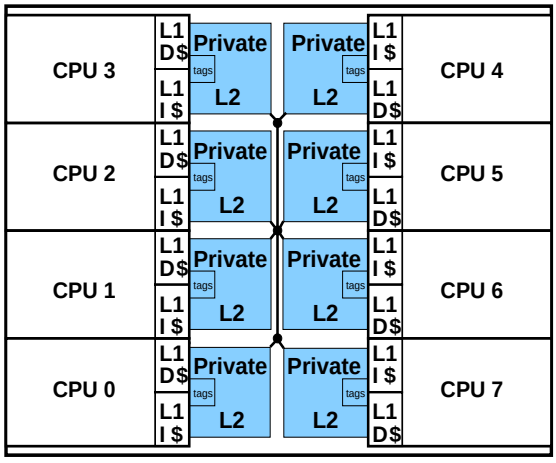


FIGURE 6-1. Private CMP Cache

reduces the mesh network's area and wire demand. To prevent bank contention from significantly affecting cache hit latency, the private L2 banks are further subbanked into 8 smaller units. Similar to the Itanium 2 microprocessor [78], each private L2 cache is closely integrated to a processor, allowing the processors to directly query their local L2 cache tags in parallel with an L1 cache access. Also having fewer cache banks cuts down on the area overhead for the cache controller logic and reduces complexity by decreasing the number of locations searched on a local cache bank miss.

6.3 Adaptive Selective Replication

ASR seeks the optimum replication level by balancing the benefits of replication against the costs. Section 6.3.1 introduces the simple memory system performance model underlying ASR and Section 6.3.2 describes ASR's replication algorithm.

6.3.1 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—previously introduced in Chapter 4—describes the average cycles for L1 cache misses normalized by instructions executed:

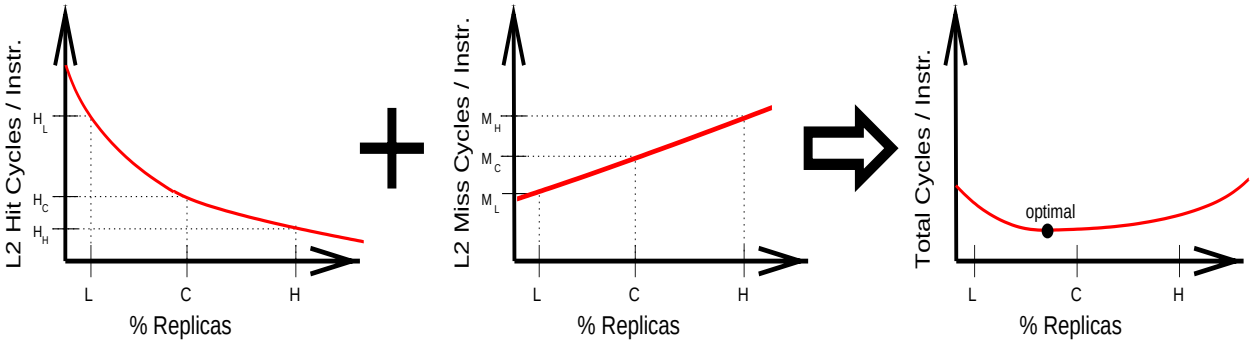


FIGURE 6-2. a) Replication Benefit

FIGURE 6-2. b) Replication Cost

FIGURE 6-2. c) Replication Effectiveness

The figures above illustrate the replication benefit, cost, and effectiveness curves. The L marks represent the next lower replication level, the C marks represent the current replication level, and the H marks represents the next high replication level.

$$\frac{\text{Private Cache L1 Miss Cycles}}{\text{Instruction}} = \frac{(P_{localL2} \times L_{localL2}) + (P_{remoteL2} \times L_{remoteL2}) + (P_{miss} \times L_{miss})}{(Instructions/L1misses)} \quad (6.1)$$

where P_x is the probability of a memory request being satisfied by the entity *x*, and *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 6-2 graphically depicts this tradeoff. The *Replication Benefit* curve, Figure 6-2a, 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 6-2b, illustrates that increasing L2 replication increases the memory cycles spent on off-chip misses. The *Replication Effectiveness* curve, Figure 6-2c, 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.

6.3.2 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 simplifies the analysis to a *local* decision of whether the amount of replication should be increased, decreased, or remain the same. Figure 6-2 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

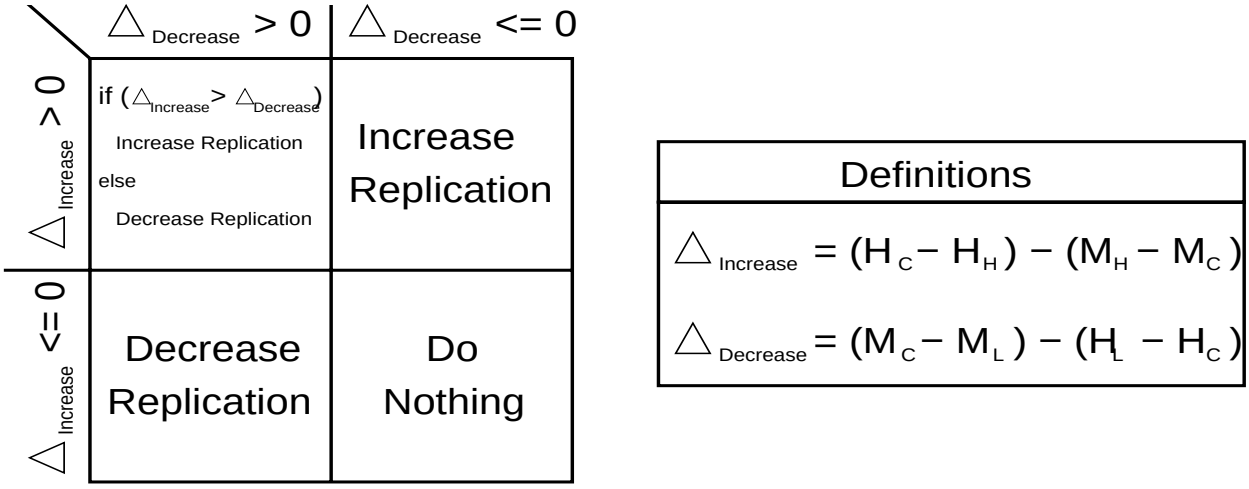


FIGURE 6-3. ASR Decision Table for Adjusting Replication

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 6-3 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.

6.4 Implementing ASR

Implementing ASR requires a CMP cache framework that supports multiple replication levels. Cooperative Caching [20] is one possibility, but this scheme requires an expensive central tag structure. Section 6.4.1 introduces the simpler Selective Probabilistic Replication (SPR) design which uses distributed state to make local replication decisions. Section 6.4.2 describes the additional hardware needed to implement ASR. Finally, Section 6.4.3 summarizes ASR's storage and energy overhead.

6.4.1 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 [76] and ring writebacks [98] 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, 111]. On an L1 cache eviction, SPR writes a shared block back to its local L2 if (i) a 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 process, SPR supports six discrete replication levels (Table 6-1). 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 [41] generates an 8-bit pseudo-random number which it compares to the current replication threshold (i.e., if random < threshold, then replicate). Like all SPR logic, the

TABLE 6-1. 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

pseudo-random number generator does not impact L2 cache access latency and is accessed only on L1 replacements. The probabilistic policy biases replications to frequently requested blocks because the most frequently requested L2 blocks are also those most frequently evicted from the L1 caches. Therefore, at the lower replication levels, SPR will devote the majority of its limited replication capacity to storing the most frequently requested shared read-only data.

6.4.2 ASR Hardware

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, ASR adds a Next Level Hit Buffer (NLHB) to each private L2 cache to track the replications of the next higher replication level. 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.

In order to approximate the lifetimes of current replica blocks, each processor's NLHB is sized to a 16 K entry, 16-way set-associative buffer. Therefore, hits to NLHB entries roughly indicate those local hits that would have been possible with the next higher replication level. In particular, for the commercial workloads, the average NLHB entry exist for 6 million cycles, which approximates the average current replica lifetime of 9 million cycles.

To reduce storage overhead, the NLHBs store only 8-bit partial tags [56] of the blocks that would have been replicated with the next higher replication level. Through exhaustive investigation, 8-bits seems to be the best tradeoff between storage overhead and reducing false positives. Specifically, 8-bits reduce each NLHB's storage cost to 16 KB, while maintaining a false positive rate below 0.05 for all evaluated workloads.

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. If a local request hits an L2 block not identified as a current replica and the block lies within the last 1 K of LRU blocks, the (M_H -M_C) counter is incremented by the off-chip memory latency.

Because precisely determining the recently used cache blocks is prohibitively expensive in hardware, ASR uses way and set counters [101] to estimate which blocks are least recently used. To reduce the storage overhead of the set counters, ASR breaks the L2 sets into 256 separate groups using the high order L2 cache index bits. By combining a L2 block's way and set-group pseudo-lru binary tree position [95], ASR determines a requested block's LRU rank. Figure 6-4 illustrates the LRU rank calculation ASR uses for a

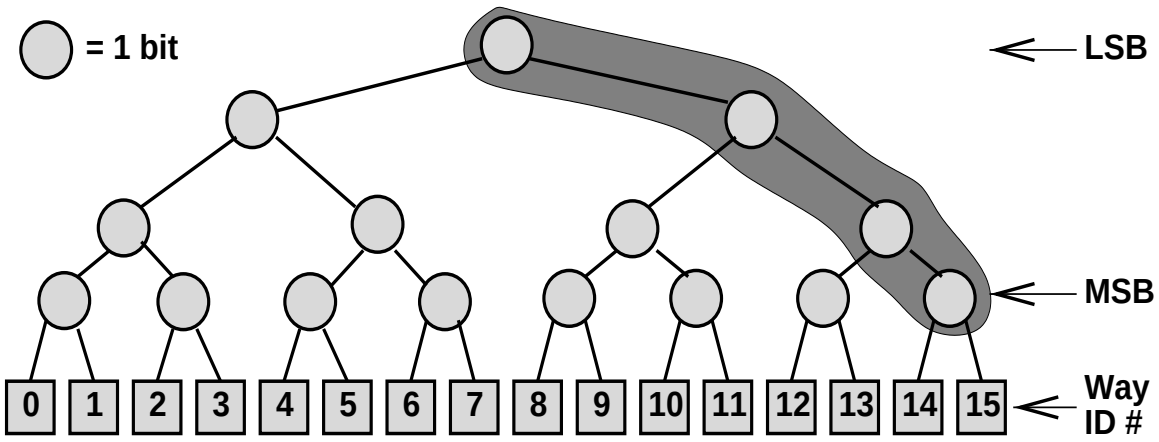


FIGURE 6-4. Binary Tree Position Translation to LRU Rank

The figure above describes how the Pseudo-LRU binary tree determines the LRU rank of each way in a 16-way cache set. By xoring a way's four associated bits (ex. the figure highlights the four bits for way 14 and 15) with a way's 4-bit identification number, a way's lru rank can be determined.

cache way's 15-bit pseudo-lru tree. The cache set-group's 255-bit pseudo-lru tree follows a similar translation.

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. 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 from the local processor 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.

To reduce storage overhead, each VTB stores 16-bit partial tags of the most recently evicted blocks in a 1 K entry 16-way set associative buffer. Similar to the NLHBs, the 16-bit partial tags reduce each VTB's storage cost to 2 KB, while maintaining a false positive rate below 0.07 for all evaluated workloads.

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. If a subsequent writeback indicates that a block would have been replicated at a lower replication level, the current replication bit is reset. 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 size monitored by the VTBs and LRU counters—1 K entries. 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 6.3.2 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.

TABLE 6-2. 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%
Consumed Area (technology generation)				~ 3 mm^2 (90 nm)	~ 1 mm^2 (45 nm)

6.4.3 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 6-2 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 6-2 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 6.6.4 directly compares ASR's storage overhead with the previous proposals [20, 26, 121].

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 [83]. 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.

6.5 Methodology

Similar to Chapter 5, we use full-system simulation based on Simics [109] and the GEMS toolset [114] to evaluate ASR against alternative cache designs. This section describes the alternative caches, system parameters, and workloads that we use in our simulation study.

6.5.1 Alternative Cache Designs

Section 6.6.4 compares ASR against two baseline configurations—shared L2 and private L2 caches—and the previous replication proposals: Victim Replication [121], CMP-NuRapid [26], and Cooperative Caching [20].

CMP-Shared. As illustrated in Figure 6-5, the CMP-Shared design assumes an 8-banked Non-Uniform Cache Architecture (NUCA) [58]. 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 average access latency.

CMP-Private. Section 6.2 previously introduced the baseline CMP-Private design (Figure 6-1). CMP-Private utilizes SPR's token broadcast protocol, which 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 [15, 48] single requestor data and replicates all shared data without the storage overhead of home blocks associated with a distributed directory protocol. How-

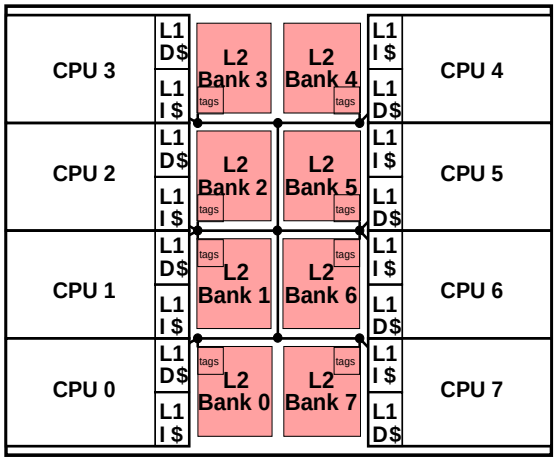


FIGURE 6-5. Layout of CMP-Shared

ever, CMP-Private does not utilize SPR's ring writeback mechanism, thus replication is unrestricted, allowing shared data replicas to 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 [121] targeted an on-chip directory protocol and statically assigned blocks home nodes (like CMP-Shared). Non-home nodes replicated blocks locally, except when a L2 cache set was filled with home blocks with remote sharers. Using a random replacement policy, non-shared home blocks were evicted before replicas. SPR-VR implements Victim Replication's replication policy by adding 1-bit per L2 cache block to identify replicas and disallowing replications when the local cache set is filled with owner blocks with identified sharers. Victim Replication's distributed directory protocol wasted 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. Though requiring more bandwidth, SPR-VR should perform strictly better than the original Victim Replication implementation because its token broadcast protocol [76] removes the home node storage overhead.

TABLE 6-3. 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 [95]	16 MB 16-way pseudoLRU [95]
avg. shared L2 / local L2 / remote L2 latency	25 / 12 / 33 cycles	44 / 20 / 52 cycles
memory latency	150 cycles	250 & 500 cycles
memory bandwidth	28 GB/s	50 GB/s

SPR-NR. CMP-NuRapid [26] maintained coherence using a shared bus and per-processor decoupled tag arrays with indirect data block pointers (6% overhead). CMP-NuRapid's replication policy allocated a local L2 tag after the first request and then locally allocated 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). The first request by a processor sets its associated bit so when that processor's subsequent requests notice the bit set, it will locally allocate the L2 block.

SPR-CC. Cooperative Caching (CC) [20] used a centralized duplicate tag structure to identify single on-chip L2 block copies, i.e. singlets, and used biased replacements to evict non-singlets first. Cooperative Caching attempts to retain globally active singlets by spilling them to a remote L2 cache. SPR-CC models the centralized tag structure using an idealized distributed tag structure.

6.5.2 System Parameters

The evaluation studies two different SPARC V9 8-processor CMP configurations targeting current and future technology. The Sun Niagara [62, 67] inspired the first CMP configuration, (the second column of Table 6-3), and the second configuration presents a CMP assuming 2010 technology [39] (column 3 of Table 6-3). The out-of-order processors have the same parameters as the out-of-order processors used in

Chapter 5. Also, the intra-chip and inter-chip protocols are the same as Chapter 5, as well as the strided prefetcher.

6.6 Evaluation

This section demonstrates ASR dynamically adapts replication to match different workload behaviors and system constraints. First, Section 6.6.1 advocates for an adaptive policy by illustrating how the optimal replication level changes depending on the workload and system configuration. Next, Section 6.6.2 shows that ASR can dynamically identify these different situations and adjust replication accordingly. Then, Section 6.6.3 illustrates how ASR balances reducing on-chip latency with minimizing off-chip misses. Finally, Section 6.6.4 compares ASR with previously proposed replication schemes and demonstrates ASR achieves robust performance across three different system configurations: the Current CMP configuration, the Future CMP configuration, and the Future CMP configuration with longer memory latency.

6.6.1 Replication Capacity and Memory Cycles

The optimal replication point shifts depending on workload behavior and CMP configuration. Figure 6-6 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 6-6a-c) presents the results for the Current CMP configuration with a 4 MB aggregate L2 cache capacity. The L2 hit cycles-per-instruction curves for the workloads: Apache, Jbb, Oltp, Zeus, and Barnes (Figure 6-6a) 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

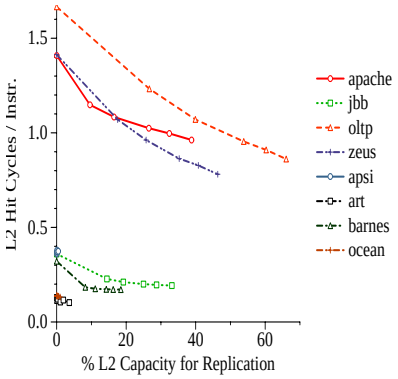


FIGURE 6-6. a) Current CMP: L2 Hit Cycles / Instr.

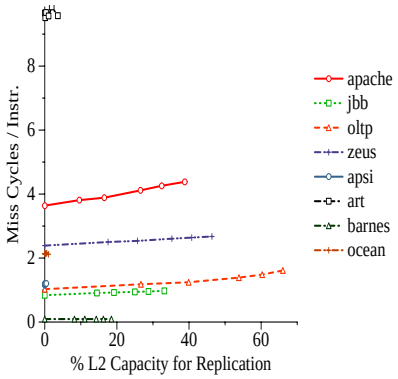


FIGURE 6-6. b) Current CMP: L2 Miss Cycles / Instr.

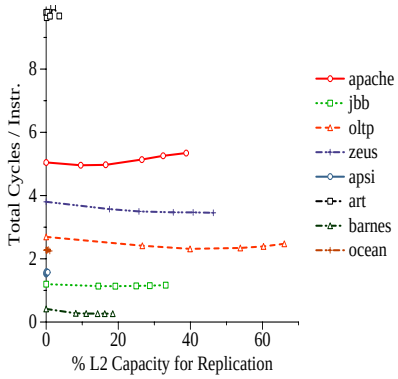


FIGURE 6-6. c) Current CMP: Total Cycles / Instr.

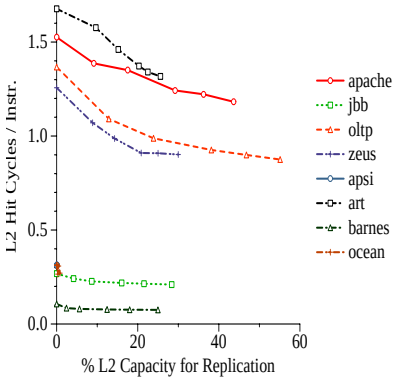


FIGURE 6-6. d) Future CMP: L2 Hit Cycles / Instr.

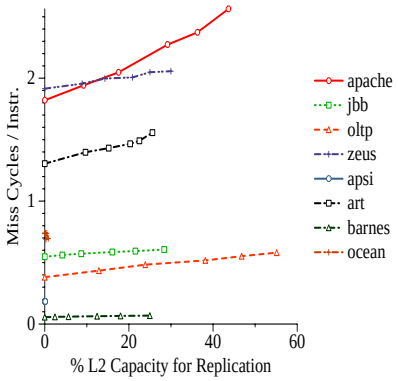


FIGURE 6-6. e) Future CMP: L2 Miss Cycles / Instr.

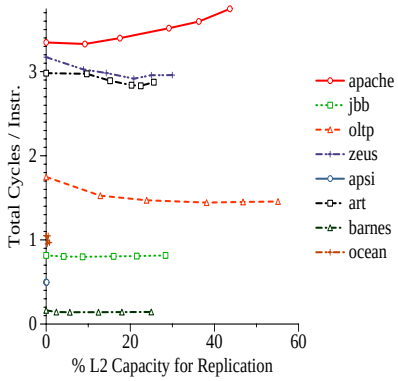


FIGURE 6-6. f) Future CMP: Total Cycles / Instr.

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 6-6b 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 0.2 for every 10% increase in replication capacity. The resulting total cycles-per-instruction curves (Figure 6-6c) 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 Ocean, 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 6-6d-f) show that the optimal level of replication changes as compared to the Current CMP configuration. For most workloads, the normalized L2 hit cycle curves (Figure 6-6d) maintain the same basic shape as those of Figure 6-6a. However, Art demonstrates how balancing replication becomes more important with larger caches [50] because its 8 MB working set (Chapter 4) now fits in Future CMP's larger cache. Figure 6-6e illustrates Future CMP's slower memory latency causes the L2 miss cycle slopes to increase with respect to those in Figure 6-6b. 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 Zeus shifted from 16% and 47%, respectively, for the Current CMP configuration, to 10% and 20% for the Future CMP configuration.

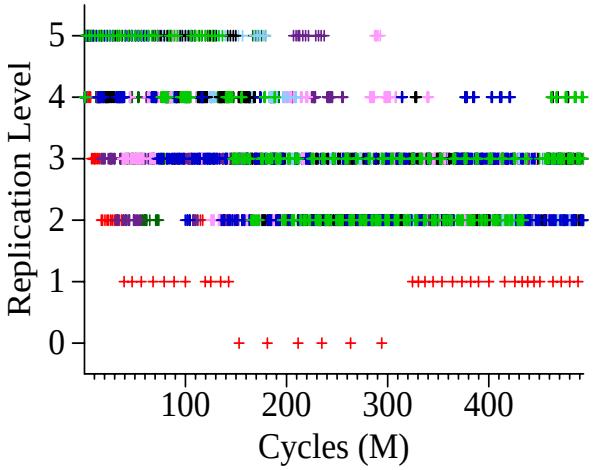


FIGURE 6-7. a) Future CMP: ASR Adaptability *Apache*

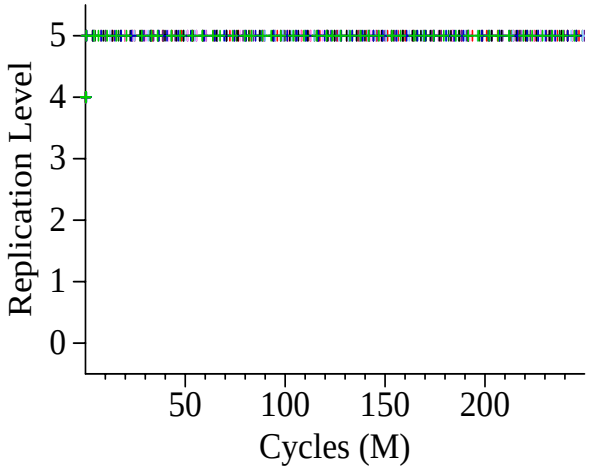


FIGURE 6-7. b) Future CMP: ASR Adaptability *Oltp*

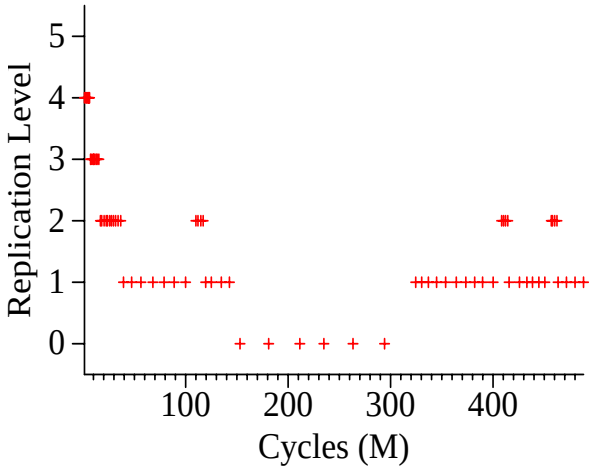


FIGURE 6-8. a) Future CMP: ASR Adaptability *Apache—Processor 0*

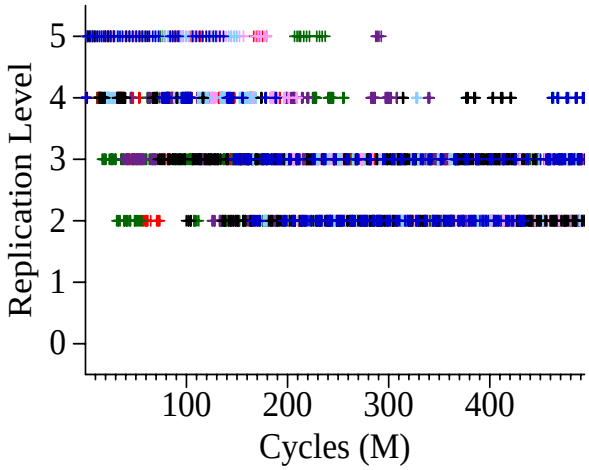


FIGURE 6-8. b) Future CMP: ASR Adaptability *Apache—Processors 1-7*

6.6.2 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 6-7 illustrates SPR-ASR's dynamic adjustment of each private L2 cache's replication level over the runtime of the workload. Both Figure 6-7a and Figure 6-7b use the future CMP configuration and each processor's replication level is initialized to level 4. Each point on the plots indicates when an SPR-ASR evaluated of its counters.

For the workload Apache (Figure 6-7a), SPR-ASR reduces the replication level to achieve the lower replication capacity preferred by the workload. In order to illustrate the benefit of ASR's local adaptability, Figure 6-8a plots the ASR level of the one processor that executes almost exclusively OS code—processor 0—and Figure 6-8b plots the ASR levels of the seven other processors. Interestingly, processor 0's ASR detects very little replication benefit and drops the replication level to levels 0 and 1 for majority of the run. In contrast, after the first 250 million cycles, the replication levels of processors 1 through 7 hover between levels 2 and 3 for the remainder of the execution. Overall, SPR-ASR dynamically identifies the improvement provided by selective replication in Apache and reduces the average L2 capacity consumed by replicas to 5%.

For the workload Oltp, which has an optimal point of replication capacity near the maximum, SPR-ASR adjusts replication to level 5 for all processors. Figure 6-7b illustrates that each processor's SPR-ASR mechanism quickly detects a benefit for replicating Oltp's large instruction footprint and moves all eight L2 caches to level 5, or 100% probability of replication, within the first 10 million cycles. The result is that on average 52% of L2 capacity is consumed by replicas.

As Figure 6-7a exemplifies, SPR-ASR can require well over a 100 million cycles to reach steady state, which equates to several hours of simulation. Therefore, in order to evaluate SPR-ASR during steady state execution, the remainder of this section initializes the replication levels to their steady state value.

6.6.3 Sharing Type Latency vs. Off-chip Misses

This subsection illustrates how SPR-ASR sacrifices shared block latency in order to reduce off-chip misses and improve performance. Similar to Chapter 5, Figure 6-9 displays the L1 miss latency to on-chip single requestor, shared read-only, and shared read-write blocks, as well as, off-chip misses. As expected, the private cache designs exploit their faster local L2 banks and reduce single requestor latency by roughly 55% versus CMP-Shared. For shared data, CMP-Private replicates all shared L2 blocks, resulting in an average

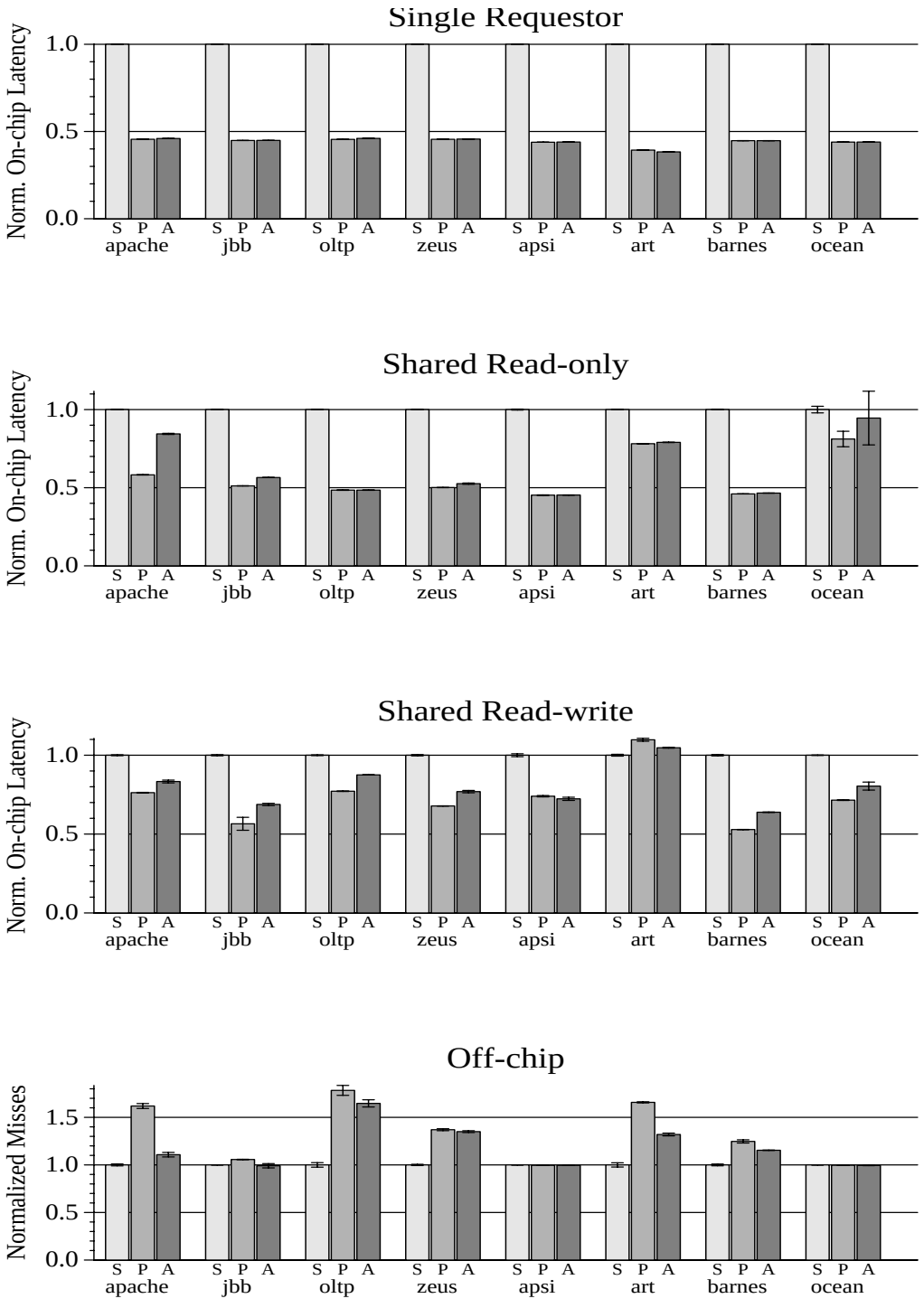


FIGURE 6-9. Future CMP: Normalized L1 Miss Latency to Sharing Types and Off-chip Misses (S: CMP-Shared, P: CMP-Private, A: SPR-ASR)

46% reduction in shared read-only latency and an average 30% reduction in shared read-write latency. In contrast, SPR-ASR limits shared read-only replication based on workload behavior and disallows shared read-write replication across all workloads. The result is SPR-ASR's Apache shared read-only latency increases by as much as 45% versus CMP-Private, and SPR-ASR's average shared read-write latency increases by 15% versus CMP-Private. Finally, Figure 6-9's off-chip miss plot displays the benefit of restricting replication. For all workloads except Apsi and Ocean, SPR-ASR encounters noticeably fewer off-chip misses than CMP-Private, with the greatest reduction being 32% for Apache. Overall, for certain workloads, SPR-ASR tradeoffs on-chip shared block latency for fewer off-chip misses. The next subsection displays how this tradeoff translates into better CMP performance.

6.6.4 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 6-10 shows the normalized runtime of each CMP design executing the eight workloads. The two SPR-CC bars represent the worst and best performing Cooperative Caching percentages[1]—we refer to these as worst and best SPR-CC. 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 0-30% versus CMP-Shared. For Oltp, SPR-ASR and best SPR-CC improve performance by 6% and 9%, respectively, versus CMP-Shared, while the other private cache designs degrade performance by 3-7%. SPR-ASR limits replication to 43% of L2 capacity, resulting in only a 43% increase in off-chip cycles versus CMP-Shared. In contrast, the other private CMP designs devote as much as 66% of L2 capacity to replicas causing at most a 99% increase in off-chip cycles versus CMP-Shared. Overall, the best performing SPR-CC percentage achieves competitive performance

1.Reminder: Cooperative Caching 100% always evicts replicas before singlets and allows all singlets to spill into one remote cache. In contrast, Cooperative Caching 0% treats replicas and singlets equally and disallows singlet spilling.

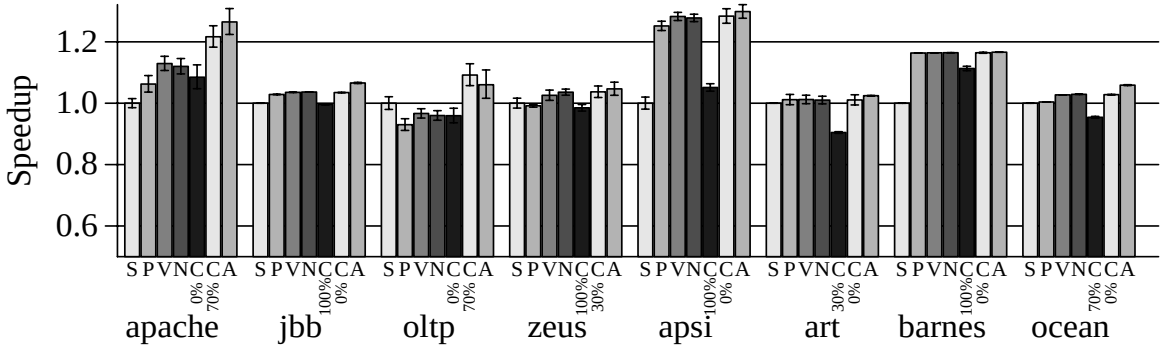


FIGURE 6-10. Current CMP: Speedups
(S: CMP-Shared, P: CMP-Private, V: SPR-VR, N: SPR-NR, C: SPR-CC, A: SPR-ASR)

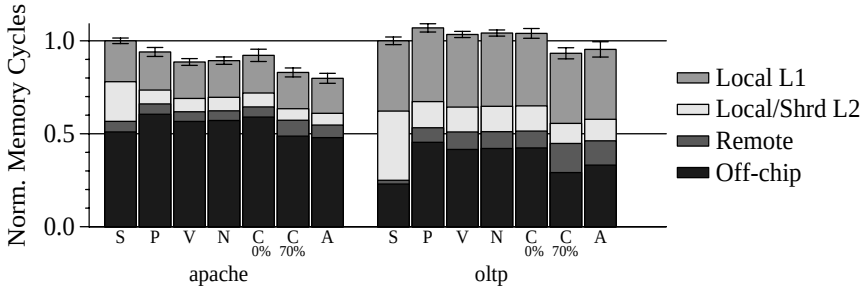


FIGURE 6-11. Current CMP: Memory Cycles
(S: CMP-Shared, P: CMP-Private, V: SPR-VR, N: SPR-NR, C: SPR-CC, A: SPR-ASR)

to that of SPR-ASR. However, the particular SPR-CC percentage varies between workloads, thus exemplifying the need for an adaptive policy.

To provider further insight, Figure 6-11 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 6-6c) 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 replication. Specifically, SPR-ASR achieves the greatest performance

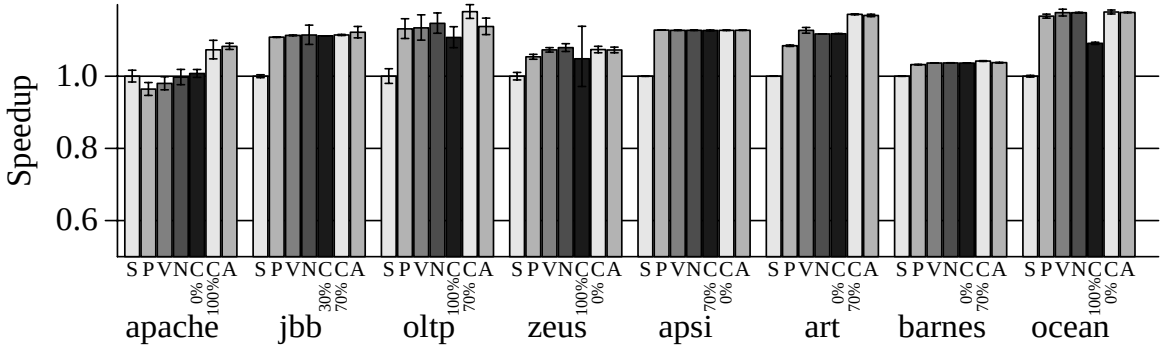


FIGURE 6-12. Future CMP: Speedups
(S: CMP Shared, P: CMP-Private, V: SPR-VR, N: SPR-NR, C: SPR-CC, A: SPR-ASR)

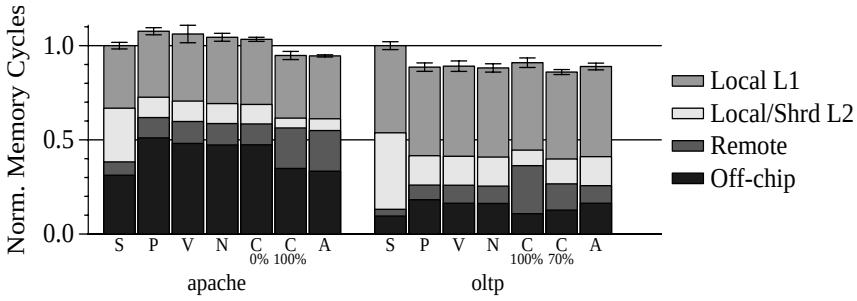


FIGURE 6-13. Future CMP: Memory Cycles
(S: CMP-Shared, P: CMP-Private, V: SPR-VR, N: SPR-NR, C: SPR-CC, A: SPR-ASR)

improvement (26% 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 6-11 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 larger cache sizes and out-of-order processors [84] change the performance tradeoff between shared and private caches. For Apache, Apsi, and Barnes, the maximum performance advantage the private cache designs exhibit over CMP-Shared diminishes to only 4-13%, while the private caches' performance advantage increases to 18% for Oltp and Ocean. Figure 6-13 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

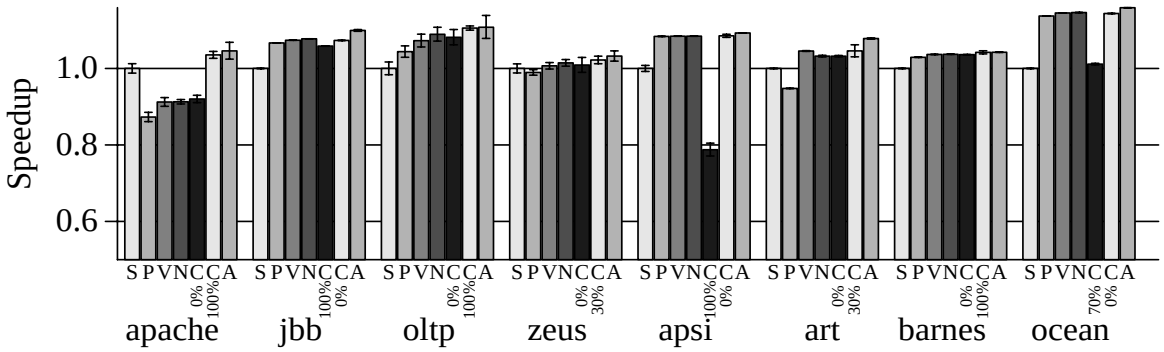


FIGURE 6-14. Future CMP *500 cycle memory latency*: Speedups (S: CMP-Shared, P: CMP-Private, V: SPR-VR, N: SPR-NR, C: SPR-CC, A: SPR-ASR)

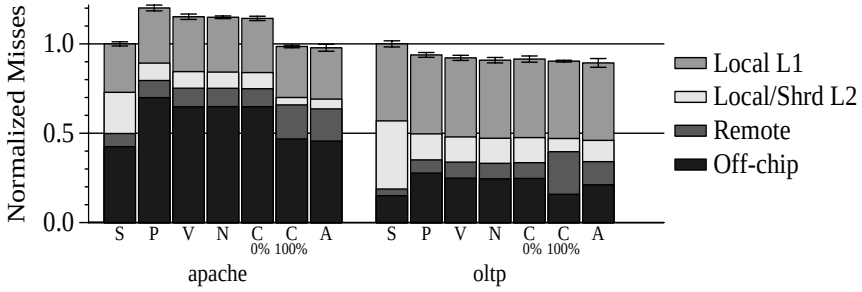


FIGURE 6-15. Future CMP *500 cycle memory latency*: Memory Cycles (S: CMP-Shared, P: CMP-Private, V: SPR-VR, N: SPR-NR, C: SPR-CC, A: SPR-ASR)

the faster private L2 caches. Oltp's frequently requested shared read-only data (Chapter 4) especially benefits from replication, enabling the private cache designs to improve performance by 13-18% versus CMP-Shared, with SPR-CC achieving the greatest improvement. However, Apache's larger working set (Chapter 4) exposes Future CMP's slower memory latency. For instance, CMP-Private and SPR-VR suffer a performance degradation versus CMP-Shared of 3% and 1%, respectively, while SPR-ASR achieves a performance improvement of 9%.

Finally, illustrates SPR-ASR also improves performance when the memory latency in the Future CMP configuration is increased to 500 cycle, thus demonstrating ASR will provide good performance within a multiple-chip CMP system with a longer memory access latency [55]. Specifically, Apache's larger working set conflicts with the 500 cycle memory latency and all private cache designs except SPR-CC and SPR-ASR suffer substantial performance degradation versus CMP-Shared. In particular, CMP-Private, SPR-VR

TABLE 6-4. 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

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%. Figure 6-15 breaks down the memory cycles for Apache and Oltp. By limiting replicas to consuming only 1% of L2 cache capacity, SPR-ASR does not suffer the off-chip cycle increase like some of other private cache designs. Instead, SPR-ASR fully utilizes its remote cache capacity, attaining 88% more remote hit cycles than CMP-Private.

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 usually performs at least as well as, 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 6-4 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 increased the aggregate L2 cache size. In comparison, CMP-NuRapid's SPR implementation adds 7-bits per L2 block—a 1-bit counter for each remote L2 cache—and Cooperative Caching's SPR implementation requires a duplicate tag per L2 cache block.

6.7 Related Work

6.7.1 Multiprocessor Memories

Much previous work has analyzed migrating and replicating data between multi-chip multiprocessors to reduce the effect off-chip latency had on performance. Some of the earliest migration work was done at the memory level in hierarchical multiprocessor networks. For instance, in the early 1990's two developed Cache Only Memory Architectures (COMA) systems used a hierarchical directory structure [40, 43] and turned the memory modules within the directory structure into logical caches, called Attraction Memories. Another study by Mizrahi *et al.* [81] evaluated the benefits of migration within switches of a hierarchical multiprocessor network. The paper advocates for migrating data directly to the processors and initiating a series of writebacks back to the root switch to create space. In contrast Kim *et al.* [58] and myself advocate for gradual migrations because multiple writebacks consume power and bandwidth.

A larger body of work exists evaluating the performance of data migration and replication assuming a flat multiprocessor network. Specifically, throughout the previous decade, significant work has compared completely hardware solutions such as CC-NUMA and Flat COMA architectures [100, 122], along with software [19, 108] and hybrid hardware/software combinations [35, 97]. The Flat COMA protocol [97, 100] removed the slow ordered network of Hierarchal COMA machines and allowed data to directly migrate towards the requesting processor on an unordered network similar to CMP-DNUCA. Another study by Verghese *et al.* [108] proposed allowing the OS to migrate and selectively replicate pages to a processor's local memory to reduce the miss penalty in the Stanford Flash CC-NUMA machine. Analogous to ASR's replication levels, their scheme utilized a "Trigger threshold" to indicate the number of misses until a page was considered hot and available for migration/replication. Similarly, Falsafi and Wood [35] utilized "Reactive Counters" on pages to determine when to relocate a page to the node's S-COMA Page Cache. Comparable to Chapter 4, Zhang and Torrellas [122] broke down working sets into three cases, Replication

data, Migration read only data, and Migrating read/write data in their performance comparison of a NUMA system with a Remote Cache (NUMA-RC) and a COMA-Flat system. Also similar to Chapter 4's run length characterization, Gupta and Weber [42] extensively analyzed invalidations patterns in a directory-based SMP. Finally, Dahlgren and Torrellas [27] provided a cohesive survey of COMA architectures.

6.7.2 Uniprocessor Caches

As previous discussed, Kim *et al.* [58] were the first to study data migration in wire-delay dominated on-chip caches. They showed migrating frequently requested data to the closer cache banks of a uniprocessor cache significantly improved performance over a typical monolithic cache. Additional researchers have proposed optimizations to the uniprocessor NUCA cache. For instance, Kodama and Sato [61] proposed a NUCA cache composed of fully associative CAMs as storage banks and proposed a pre-promotion scheme to prefetch the i+1 block after a request for block i. The extra freedom of movement provided by the fully-associative 128 KB banks (with 5 cycle bank access time) enabled significant performance improvement, but the power implications were not evaluated. Also, Foglia, Mangano, and Prete [36] proposed two D-NUCA variations, called the Triangular D-NUCA caches, that were optimized for embedded applications. By exploiting D-NUCA's triangular access behavior, Foglia *et al.* reduced the NUCA cache size—and it's static power—by two, while achieving most of the performance of the rectangular D-NUCA cache.

By utilizing indirect tag pointers to access the data array, Chishti, Powell, and Vijaykumar [25] further improved block migration performance in a uniprocessor cache. The extra level of indirection provided by NuRapid's indirect tags, reduced conflicts in the nearest cache banks, allowing more of the critical working set to be located in the closest banks.

In contrast to these uniprocessor studies, this dissertation studies migration and replication within a CMP cache.

6.7.3 Chip Multiprocessor Caches

Recently, researchers have evaluated the effectiveness of migration and replication within CMP caches. In particular, some researchers have proposed using migration and replication within a shared CMP cache by dynamically changing its logical mapping. For instance, Liu, Sivasubramaniam, and Kandemir [70] evaluated dynamically remapping within a bus-based CMP. Specifically, their focus was to reduce cache trashing between processors operating on different data, while allowing processors accessing the same data to share storage. They utilized profile information to determine the optimal cache configuration for each dynamic unit of time, called an epoch, and their scheme achieved significant performance improvement versus the baseline shared and private CMP cache design. Another, similar remapping proposal by Huh *et al.* [48], investigated dynamic partitioning within the CMP-DNUCA cache design. They confirmed results that data sharing between processors limits migration's performance benefit with the CMP-DNUCA cache. To combat this problem, they proposed logically sharing only subsections of cache banks between processors.

Complementary to the CMP-DNUCA remapping proposal, other researchers have improved CMP-DNUCA using bloom filters or innovative 3D integration technology. For example, Ricci *et al.* [92] significantly reduce CMP-DNUCA's "smart search" mechanism storage overhead by replacing the centralized 6-bit partial tag structure—1.5 MB—with 128 distributed bloom filters—160 KB. A set of 16 1-bit entry bloom filters is managed locally by a particular processor and a bloom filter is cleared once its false positive rate exceeds 50%. Ricci *et al.* show that this bloom filter design achieves better than 85% accuracy for a set of scientific workloads, but their scheme had to deal with false negatives. Another study by Li *et al.* [69], demonstrated migration performs significantly better within a novel three-dimensional L2 cache than a typical two-dimensional L2 cache because more cache banks can be integrated closer to more processors. While three-dimensional integration adds costly manufacturing steps, its multi-dimensional locality properties increase the importance of migration and replication.

Other researchers have analyzed migration and replication within private CMP cache hierarchies. For example, Harris's Synergistic Caching [44] investigated logically grouping L1 caches in a 64-processor CMP to exploit inter-processor sharing. Harris proposed three cache block movement modes: beg, borrow, and steal, which are similar to three block movements we analyzed: replicating blocks between multiple cache banks, writing back blocks to remote cache banks, and migrating blocks to the requesting cache bank. Because no single mode was always the best, Harris advocated for an adaptive configuration mechanism, but did not evaluate such a mechanism. Another study by Yeh and Reinman [118] proposed the Performance Driven Adaptive Sharing (PDAS) cache architecture for embedded systems. PDAS dynamically adapted cache partitioning within a private CMP cache hierarchy to improve multiprogrammed workload quality-of-service. PDAS's software algorithm utilized hardware counters and ran for 10,000 cycles every 100 millionth interval. In contrast, ASR's algorithm is simple enough for hardware and is focused on multithreaded workload performance.

Other recent work focused on operating system interaction within CMP caches. For instance, Kim, Chandra, and Solihin [59] analyzed fair CMP cache sharing and partitioning while executing a mixture of co-scheduled threads within a multiprogrammed environment. While their technique focused avoiding OS scheduler problems, they did propose a hardware cache monitoring scheme similar to ASR. The important differences are their algorithm strictly concentrates on cache storage allocation and relies on profile information. Similarly, Chandra *et al.* [18] proposed three models to predict the L2 miss rates when co-scheduled threads share the L2 cache and Suh, Devadas, and Rudolph [101] proposed the Stack Distance Competition (SDC) model that used LRU distance hit counters to estimate how working sets will merged together. Both envision their models can help guide the operating system scheduler.

Analogous to ASR's replication cost monitoring, Zhang and Asanovic [120] used a miss tag buffer to track what cache misses could have been hits if the cache was full sized. The difference is Zhang *et al.*'s miss tag buffer stored full size tags and was used to save energy in a automatically resizable cache.

Similar to SPR's ring writeback mechanism, Speight *et al.* [98] presented two mechanisms to manage cache hierarchies in a Power4-like CMP connected with a ring intra-chip connection network. The first mechanism, Write Back History Table (WBHT), saved bandwidth on the intra-chip network by not writing back clean L2 data when the L2 believed that the L3 already has a copy. The second mechanism, the L2 Snarf Table, reduced latency by deciding when to keep critical blocks in other fast access L2 caches instead of writing them back to the slower L3 cache. The difference between these mechanisms and SPR's ring writeback mechanism is that SPR uses the ring to avoid replication instead of to save latency.

The most closely related proposals to Adaptive Selective Replication are the previously discussed Victim Replication [121], CMP-NuRapid [26], and Cooperative Caching [20] 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.

6.8 Summary

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. This chapter proposes 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 rep-

lication. By performing an opportunity analysis, ASR adjusts the degree of replication to match the current workload behavior and system configuration. This chapter showed ASR usually performs as well as the best alternative design and improves performance for commercial workloads with large working sets.

Chapter 7

Transmission Line Caches

This chapter investigates how transmission lines can improve CMP cache performance. Specifically, the chapter demonstrates transmission lines consistently improve performance for both shared and private caches. This chapter also shows that transmission lines can work in concert with techniques, such as Adaptive Selective Replication, to improve performance beyond what can be achieved by either technique alone.

The chapter begins with Section 7.1 motivating the benefits of using on-chip transmission lines with CMPs. Next, Section 7.2 and Section 7.3 evaluate how transmission lines improve a shared and private cache hierarchy, respectively. Then, Section 7.4 compares the performance of a shared transmission line cache with the performance of a private transmission line cache combined with ASR. Finally, Section 7.5 summarizes related work and Section 7.6 concludes.

7.1 Motivation

By utilizing on-chip transmission lines, Transmission Line Caches (TLCs) can substantially improve upon conventional cache designs. As previously described in Chapter 2, on-chip transmission lines reduce communication latency by an order of magnitude versus conventional wires. Additionally, transmission lines can travel long distances without repeaters, thus allowing TLCs to implement more efficient cache layouts that are not possible with standard repeated wires. For instance, the shared CMP-TLC design (Shared CMP-TLC), described in Section 7.2, provides fast access to L2 cache banks located on a CMP's perimeter, while also facilitating fast L1 cache transfers between centrally located processors. However, on-chip transmission lines sacrifice considerable bandwidth density because they require very thick wires and

intermetal dielectrics that are only available in a chip's uppermost metal layers. Thus, TLCs can be significantly faster and more efficient than traditional caches, but restricted transmission line bandwidth may limit their overall performance benefit.

TLC's performance depends on both the latency and bandwidth of transmission lines. Previously, the high-level model in Chapter 3 showed that under low contention, a 64-banked cache utilizing thin-fast wires attains significantly lower average cache access latency than a cache using wide-slow wires. Using full system simulation, Section 7.2 confirms those results and shows that TLC designs can attain up to a 15% performance improvement over a shared 64-banked cache using conventional wide-slow wires. However, for the 64-banked Shared CMP-TLC design, attaining this sizeable speedup requires abundant bandwidth from a global on-chip network exclusively composed of transmission lines.

In contrast, the 8-banked Private CMP-TLC design requires less transmission line bandwidth because it efficiently combines all three latency management techniques. Unlike Shared CMP-TLC, Private CMP-TLC does not rely on an exclusive transmission line network to enable its layout. Instead, Private CMP-TLC places its private cache banks in the die's center so its interconnection network can combine a 2-D mess of conventional wires with point-to-point transmission lines. In particular, Section 7.3 demonstrates communicating only request messages across transmission lines and sending data over conventional wires can achieve 95-99% of the performance attained by an exclusive transmission line network with 20 times the link bandwidth. Also, Section 7.3 shows Private CMP-TLC combines migration, replication, and transmission lines to reduce single requestor, shared read-only, and shared read-write latency, respectively. Furthermore, Section 7.3 shows that ASR leverages Private CMP-TLC's lower latency to reduce off-chip misses. Finally, Section 7.4 illustrates Private CMP-TLC achieves equivalent average performance to Shared CMP-TLC despite using four times fewer transmission lines and slower cache banks.

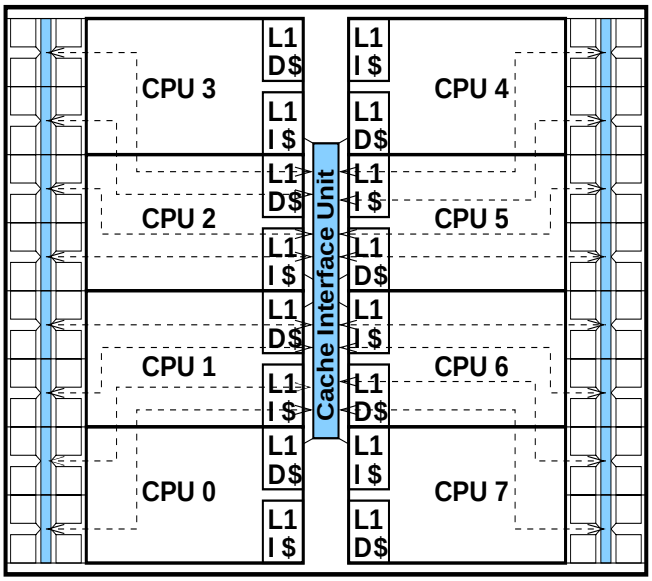


FIGURE 7-1. Shared CMP-TLC

7.2 Shared CMP-TLC

7.2.1 Overview

The Shared CMP-TLC design (Figure 7-1) not only provides fast access to the entire shared L2 cache by utilizing transmission lines, but also facilitates fast L1 cache transfers. Shared CMP-TLC's unique layout places the processors in the center of the die so that all processors have fast access to the Cache Interface Unit (CIU) that communicates with the 64 L2 cache banks on the periphery of the die. Another secondary advantage of placing the processors in close proximity to each other is it minimizes the impact of global wire delay on remote L1 cache hit latency. Because transmission lines do not require repeaters, Shared CMP-TLC creates a fast, direct connection between the centrally located CIU and the peripherally located storage arrays by routing directly over the processors. Four banks (2 adjacent groups of 2 banks) share a common pair of thin 8-byte wide unidirectional transmission line links to the CIU. To mitigate the contention for the thin transmission line links, the Shared CMP-TLC design provides 16 separate links to differ-

TABLE 7-1. Shared CMP-TLC Cache Interface Unit Height Breakdown

Link Type	Total Links Used	Total Links per Side	Total Side Width
TL 9 mm	4	2	1.28 mm
TL 10 mm	4	2	1.54
TL 11 mm	4	2	1.79
TL 13 mm	4	2	2.05
Cache Interface Unit Total Height (assuming 0.67 track utilization)			9.94

ent segments of the L2 cache. These links provide sufficient bandwidth for most workloads, but for certain workloads they become a bottleneck.

For those workloads that require significant bandwidth, Wave Division Multiplexing (WDM) can increase the effective bandwidth of transmission lines. Currently, the on-chip transmission lines implemented in test chips perform no bit multiplexing [21, 52]. Assuming no multiplexing, Shared CMP-TLC can only use 8-byte wide links before requiring multiple dedicated interconnect metal layers. However, in the future, on-chip optical communication using a polymer waveguide layer and Wave Division Multiplexing (WDM) [23] could achieve 80-byte wide links with similar manufacturing costs. Therefore, this section explores the architectural ramifications WDM technology may have on Shared CMP-TLC by also analyzing Shared CMP-TLC with 80-byte wide links—referred to as Shared CMP-TLC-WDM.

The size of the centralized CIU significantly impacts the performance of Shared CMP-TLC. The widths of the transmission lines used in Shared CMP-TLC determine CIU's size. In 45nm technology, the CIU must be approximately 1 cm tall to accommodate all the transmission lines. Using the transmission line links specified in Table 2-1 of Chapter 2, Table 7-1 shows that the total height of Shared CMP-TLC's CIU is 9.94 mm assuming a track utilization efficiency of 0.67 [39]. To reduce contention, the Shared CMP-TLC cache interface uses 80-byte wide conventional wires to communicate between the transmission line transceivers located on its edges and the eight processor access points. Overall, messages encounter 2-10 cycles of communication latency within the cache interface unit depending on their source and destination.

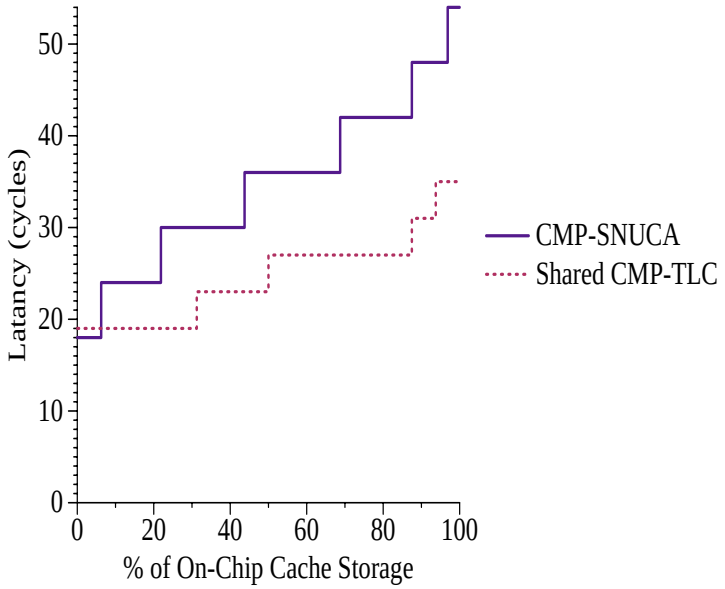


FIGURE 7-2. Uncontended Latency Comparison Between CMP-SNUCA and Shared CMP-TLC

Figure 7-2 compares the uncontended L2 cache hit latency between the 64-banked CMP-SNUCA (Figure 7-3) design introduced in Chapter 5 and Shared CMP-TLC. The plotted hit latency includes L1 miss latency, i.e. it plots the load-to-use latency for L2 hits. While Shared CMP-TLC achieves a much lower average hit latency than CMP-SNUCA, CMP-SNUCA exhibits slightly lower latency to the closest 1 MB to each processor. For instance, Figure 7-2 shows all processors in the CMP-SNUCA design can access their local bankcluster (6.25% of the entire cache) in 18 cycles or less. Previously, in Chapter 5, CMP-DNUCA attempted to maximize the hits to this closest 6.25% of the NUCA cache through migration, while Shared CMP-TLC utilizes a much simpler logical design and provides fast access for all banks.

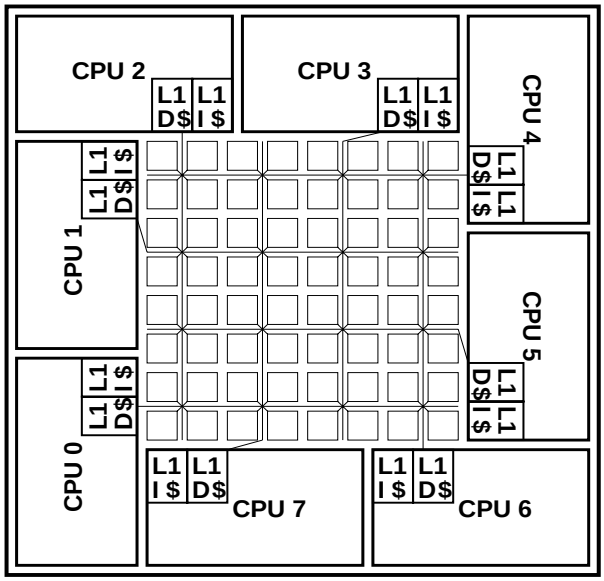


FIGURE 7-3. CMP-SNUCA

7.2.2 Methodology

Similar to Chapter 5 and Chapter 6, this chapter uses full-system simulation based on Simics [109] and the GEMS toolset [114] to evaluate Shared CMP-TLC. The only differences is the on-chip transmission lines input buffers are extended to hold 10 entries. This is because the limited bandwidth provided by these links results in greater contention.

To fully take advantage of Shared CMP-TLC's fast data transfers between L1 caches, the token broadcast protocol, previously evaluated in Chapter 5 and Chapter 6, must be altered. Specifically, by broadcasting local L1 misses instead of relying on the L2 cache to forward requests to other on-chip L1 sharers, the token protocol allows direct L1 cache-to-cache transfers at the cost of requiring more L1 cache snoop bandwidth. Similarly, a traditional broadcast snooping protocol [22] would provide comparable latency characteristics at the cost of implementing the Shared CMP-TLC's cache interface unit as an ordered crossbar.

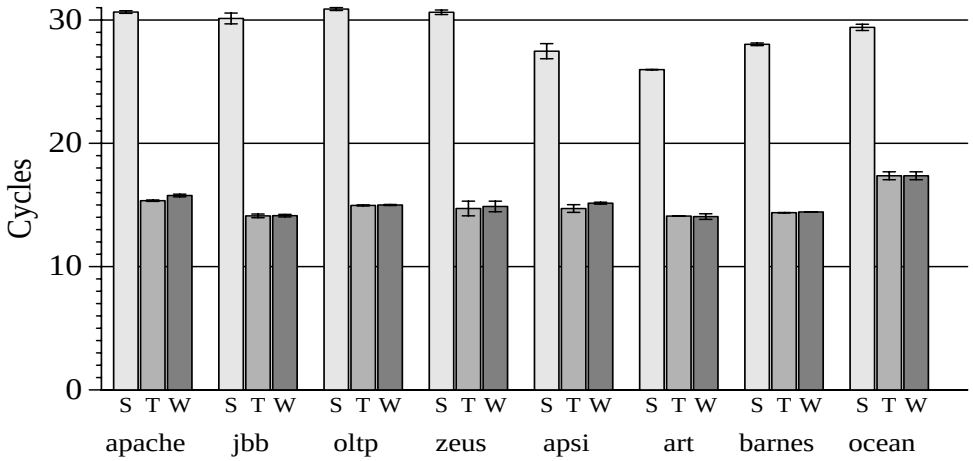


FIGURE 7-4. Shared CMP-TLC*:* Average Remote L1 Cache Hit Latency (S: CMP-SNUCA, T: Shared CMP-TLC, W: Shared CMP-TLC-WDM)

7.2.3 Evaluation

Shared CMP-TLC's combination of fast L1 cache transfers and fast access to its L2 banks allows it to significantly reduce on-chip latency versus the baseline CMP-SNUCA design. Figure 7-4 compares the average remote L1 hit latency of Shared CMP-TLC and Shared CMP-TLC-WDM to CMP-SNUCA and shows that the Shared CMP-TLC designs roughly cut remote L1 hit latency in half. Since both Shared CMP-TLC designs use the same high-bandwidth cache interface unit, both designs achieve virtually the same remote L1 latency.

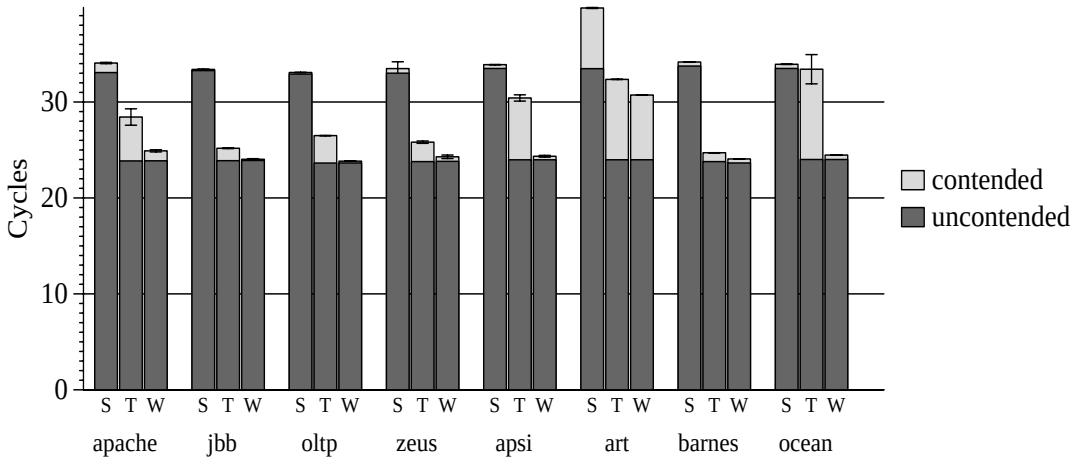


FIGURE 7-5. Shared CMP-TLC*:* Average L2 Cache Hit Latency (S: CMP-SNUCA, T: Shared CMP-TLC, W: Shared CMP-TLC-WDM)

Figure 7-5 plots the average L2 hit latencies for the three designs and illustrates that the L2 latency reduction provided by Shared CMP-TLC depends on transmission line bandwidth. The high-bandwidth networks of CMP-SNUCA and Shared CMP-TLC-WDM provide sufficient bandwidth and largely eliminate contention. In particular, for all workloads except Art, CMP-SNUCA and Shared CMP-TLC-WDM incur no more than one cycle of delay due to contention. Conversely, the limited bandwidth of Shared CMP-TLC results in contention that adds 1-9 delay cycles to L2 hits with Apsi and Ocean exhibiting the highest delay. Finally, for Art, all three designs encounter at least 8 extra contention cycles. However, these cycles are not due to bandwidth contention, but rather protocol contention. Specifically, for Art, processors frequently issue simultaneous requests for the same block, causing many of these requests to be queued up at the L2 cache controller.

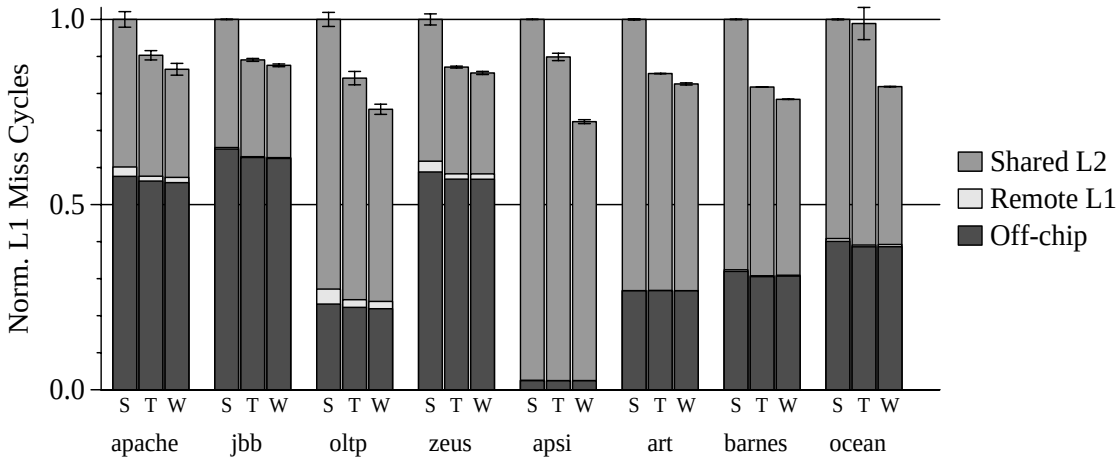


FIGURE 7-6. Shared CMP-TLC: L1 Miss Cycles Breakdown (S: CMP-SNUCA, T: Shared CMP-TLC, W: Shared CMP-TLC-WDM)

Though Shared CMP-TLC reduces remote L1 latency far greater than L2 latency, it's L2 latency reduction accounts for most of it's performance improvement. Figure 7-6 breaks down the cycles spent on L1 misses into three categories: the average memory access time contributed by L2 hits 'Shared L2', by remote L1 hits 'Remote L1', and by Off-chip misses 'Off Chip'. The Shared L2 bars display that Shared TLC's L2 latency reduction supplies most of its benefit. As forecast by Figure 7-5, the reduction of L2 hit cycles depends on the transmission line bandwidth, with the scientific workloads Apsi [6] and Ocean [94] exhibiting the greatest dependence on bandwidth because their data streaming behavior. Meanwhile, Shared CMP-TLC not only reduces Remote L1 cycles, but Off Chip bars indicate that Shared CMP-TLC reduces off chip cycles by reducing the on-chip latency between the L2 cache banks and the memory interface.

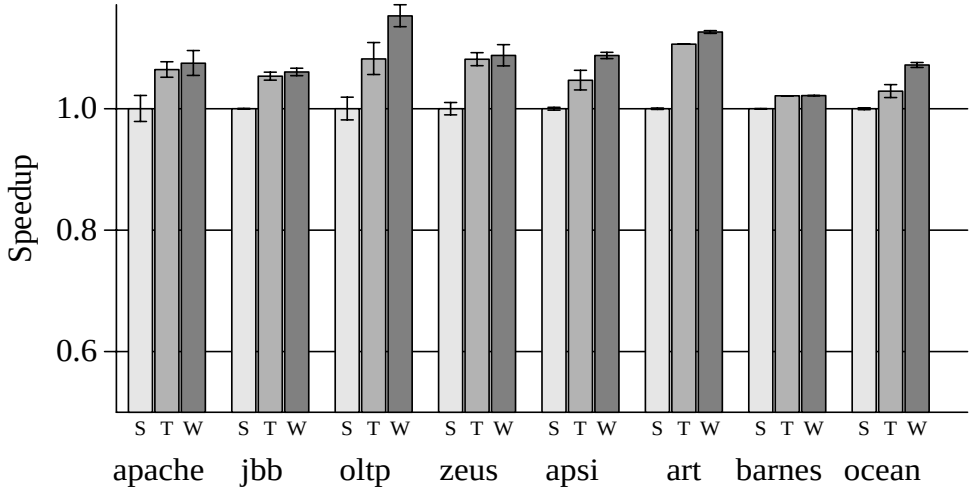


FIGURE 7-7. Shared CMP-TLC: Speedup
(S: CMP-SNUCA, T: Shared CMP-TLC, W: Shared CMP-TLC-WDM)

Overall, Shared CMP-TLC consistently outperforms the baseline CMP-SNUCA design, but bandwidth contention prevents Shared CMP-TLC from achieving its full performance potential. Specifically, Figure 7-7 shows that Shared CMP-TLC and Shared CMP-TLC-WDM improve performance by 2-8% and 2-15% versus CMP-SNUCA, respectively. The workloads that show the greatest performance disparity between Shared CMP-TLC and Shared CMP-TLC-WDM are Oltp, Apsi, and Ocean. Apsi's and Ocean's performance disparity is caused by the previously mentioned bandwidth contention issues. Meanwhile, Oltp's performance disparity is attributed to the fact that Shared CMP-TLC-WDM reduces L2 hit latency by 3 cycles versus Shared CMP-TLC (Figure 7-5), which leads to a 9% reduction in performance critical L1 instruction cache miss latency [89] versus Shared CMP-TLC.

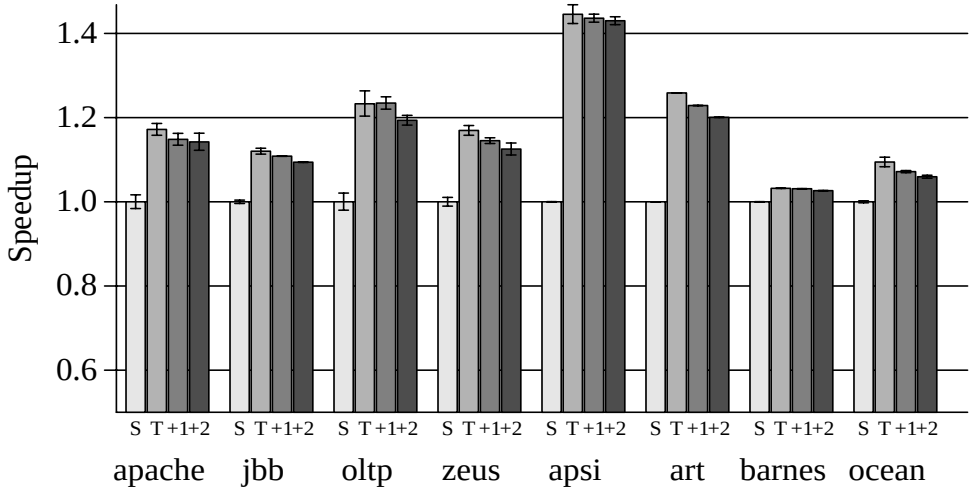


FIGURE 7-8. Shared CMP-TLC *Transceiver Sensitivity*: Speedup (S: CMP-SNUCA, T: Shared CMP-TLC, +1: Shared CMP-TLC with one extra transceiver delay cycle, +2: Shared CMP-TLC with two extra transceiver delay cycles)

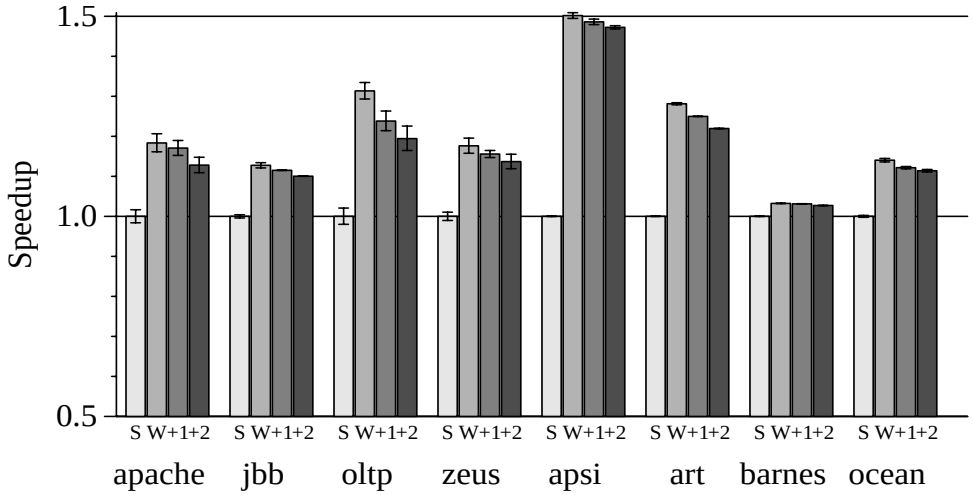


FIGURE 7-9. Shared CMP-TLC-WDM *Transceiver Sensitivity*: Speedup (S: CMP-SNUCA, W: Shared CMP-TLC-WDM, +1: Shared CMP-TLC-WDM with one extra transceiver delay cycle, +2: Shared CMP-TLC-WDM with two extra transceiver delay cycles)

Transceiver Delay Sensitivity Analysis. Based on the transmission line transceivers implemented in test chips [21, 52], the previous subsections assumed single-cycle transmission line transceiver delay. However, due to increasing noise susceptibility or the integration of WDM, future transmission line transceiver circuits may require multiple cycle delays. Therefore, this subsection analyzes Shared CMP-TLC's sensitivity to transceiver delay. Figure 7-8 and Figure 7-9 compare the performance of single-cycle trans-

ceivers to transceivers with an additional one and two cycles of delay for Shared CMP-TLC and Shared CMP-TLC-WDM, respectively. As expected, the extra transceiver delay causes consistent performance loss, with all workloads except Barnes experiencing noticeable performance degradation. For Shared CMP-TLC, one extra cycle of transceiver delay results in an average 1% performance degradation and two extra cycles results in an average 3% performance degradation. Where as for Shared CMP-TLC-WDM, one extra transceiver delay cycle results in an average 2% performance degradation and two extra cycles results in an average 4% performance degradation. Thus, transmission lines' benefit not only relies on fast wires, but also fast transceivers, especially when bandwidth contention is decreased.

7.3 Private CMP-TLC

7.3.1 Overview

On-chip transmission lines also improve private CMP cache performance, but the improvement is substantially less than for the shared CMP cache. Though the Private CMP-TLC design doesn't achieve as impressive of a performance improvement over its baseline as Shared CMP-TLC, Private CMP-TLC can effectively use thinner transmission lines. Furthermore, Private CMP-TLC can utilize the other two latency management techniques, migration and selective replication, to perform competitively with Shared CMP-TLC. This section begins with the description of Private CMP-TLC. Later the section discusses how transmission lines and Adaptive Selective Replication (ASR) interact to provide better performance than what is possible by either technique alone.

The Private CMP-TLC design improves performance over the baseline private cache design by using transmission lines to reduce remote cache hit latency. Identical to the private CMP cache evaluated in Chapter 6, Private CMP-TLC (Figure 7-10) splits the L2 into eight private 2 MB caches and allows each processor to lookup the L2 cache tags in parallel with an L1 cache accesses [49]. When a request misses in a local pri-

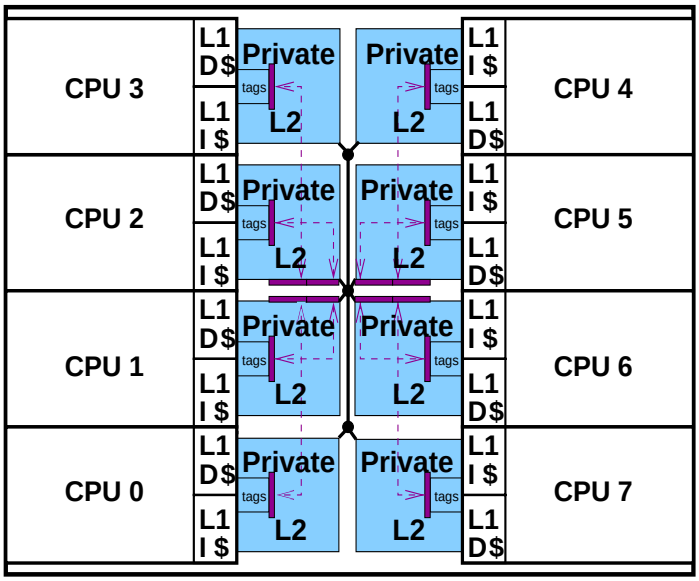


FIGURE 7-10. Private CMP-TLC

vate L2 cache, the request is broadcast across the transmission line network to the other remote L1 and L2 caches. Specifically, transmission lines decrease remote L1 and L2 cache hit latency from 33 and 52 cycles, respectively, for the baseline to 23 and 42 cycles, respectively, for Private CMP-TLC.

Private CMP-TLC reduces transmission line network overhead versus Shared CMP-TLC. In particular, eight 8-byte wide transmission lines connect each of the eight private L2 cache banks with a centralized switch. In comparison, the 64-banked Shared CMP-TLC design required 16 8-byte wide links to connect with its more distributed 64-banked shared cache. Furthermore, the maximum distance between Private CMP-TLC's L2 cache tags and the center switches is shorter (approximately 9 mm) than the maximum transmission line distance in Shared CMP-TLC (approximately 13 mm). The result is Private CMP-TLC's transmission line network consumes 62% less substrate area and 69% less interconnect area than Shared CMP-TLC's transmission line network.

Private CMP-TLC can further reduce transmission line overhead by combining a thin 4-byte wide transmission lines network exclusively dedicated to request messages with a conventional network used for sending all other messages. Unlike Shared CMP-TLC, Private CMP-TLC doesn't rely on transmission

lines to route over large on-chip processors. Instead, Private CMP-TLC routes the transmission lines over centrally located L2 cache banks. Since the L2 cache banks are smaller and more regular shaped, a 2-dimensional mesh interconnect using conventional wires can be placed between the L2 banks underneath the transmission line network. The combination network allows large 72-byte wide writeback and response messages to be off-loaded onto a high-bandwidth conventional wire network and only requests are broadcast across the transmission line network. This design is referred to as Private CMP-TLC-Request.

Though Private CMP-TLC reduces transmission line overhead versus Shared CMP-TLC, transmission line bandwidth contention still prevents Private CMP-TLC from reaching its full performance potential. Similar to Shared CMP-TLC, Private CMP-TLC is also evaluated using the 80-byte wide links possible with WDM [23]. Because only private L2 misses access the transmission line network, the performance impact of wider transmission line links is less than Shared CMP-TLC. However, those workloads that significantly utilize remote caches or have bursty request behavior, the wider links still provide noticeable improvement.

Finally, Private CMP-TLC can be amended with other latency tolerant techniques, such as Adaptive Selective Replication (ASR), to provide better performance than what is possible from either technique alone. Previously, Chapter 6 demonstrated ASR can adaptively restrict replication and sacrifice higher shared block latency for fewer off-chip misses. By dynamically monitoring workload demand, ASR only restricts replication when fewer off-chip misses is more beneficial than higher on-chip latency. Transmission lines reduce the cost of remote cache hits and the overall impact of on-chip latency. Therefore, ASR can use transmission line to not only decrease on-chip latency, but ASR can also leverage transmission lines' lower remote latency to further reduce off-chip misses beyond what is beneficial when using conventional wires.

7.3.2 Methodology

The methodology used to evaluate the Private CMP-TLC design matches that of Section 7.2. This includes the 10-entry transmission lines input buffers used to isolate queuing delay from the thin transmission line

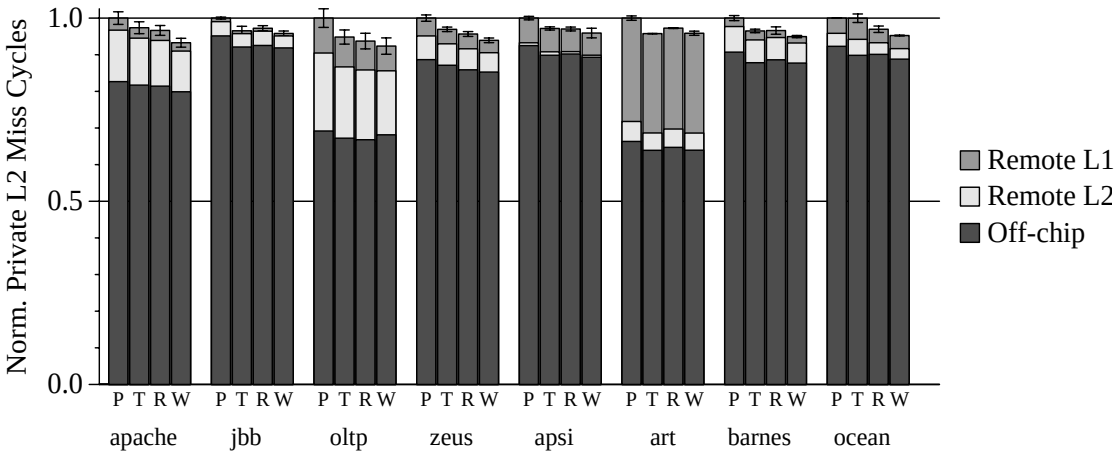


FIGURE 7-11. Private CMP-TLC: Private L2 Miss Cycles Breakdown (P: CMP-Private, T: Private CMP-TLC, R: Private CMP-TLC-Request W: Private CMP-TLC-WDM)

links. The cache coherence protocol utilized by Private CMP-TLC is the same token protocol used in Chapter 6. While requiring more bandwidth, the token protocol removes the home node storage overhead of a distributed directory protocol [121] and the duplicate tag storage overhead centralized directory protocol [20]. Furthermore, the token protocol allows Private CMP-TLC to fully exploit the fast point-to-point connections between the on-chip processors.

7.3.3 Evaluation: Baseline Private CMP Protocol

Due to the low percentage of remote cache hits, transmission lines provide only marginal benefit to the baseline CMP-Private cache. Figure 7-11 breaks down the cycles spent on private L2 misses into three categories: the average memory access time contributed by remote L1 hits ‘Remote L1’, by remote L2 hits ‘Remote L2’, and by Off-chip misses ‘Off Chip’. The Private CMP-TLC designs attain a 0-7% reduction in private L2 miss cycles. The three commercial workloads that exhibit the highest remote cache usage—Apache, Oltp, and Zeus—encounter a larger cycle reduction with the higher bandwidth networks—Private CMP-TLC-Request and Private CMP-TLC-WDM. Also the scientific workload Ocean encounters a large cycle reduction with the higher bandwidth network because of its bursty request behavior.

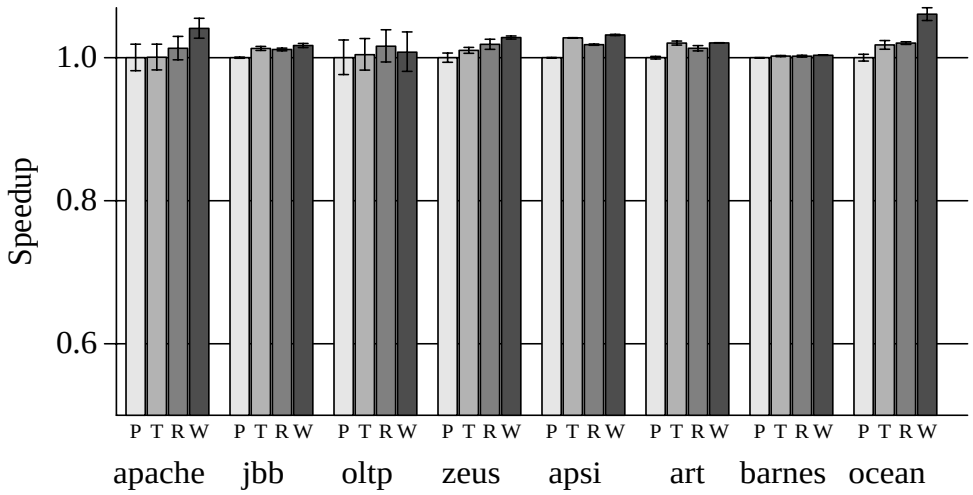


FIGURE 7-12. Private CMP-TLC: Speedup
(P: CMP-Private, T: Private CMP-TLC, R: Private CMP-TLC-Request
W: Private CMP-TLC-WDM)

Figure 7-12 compares the performance of the Private CMP-TLC designs to that of baseline private CMP design and illustrates that for all workloads except Ocean, Private CMP-TLC improves performance by less than 4%. Interestingly, for six of the eight workloads, Private CMP-TLC-Request matches or exceeds the performance of Private CMP-TLC despite the fact that it uses transmission line links of half the width. Private CMP-TLC-Request's comparable performance exemplifies that even very narrow transmission line links can improve performance if they target critical low-bandwidth communication. Overall, the average performance improvement provided by Private CMP-TLC, Private CMP-TLC-Request, and Private CMP-TLC-WDM is 1%, 1%, and 3%, respectively.

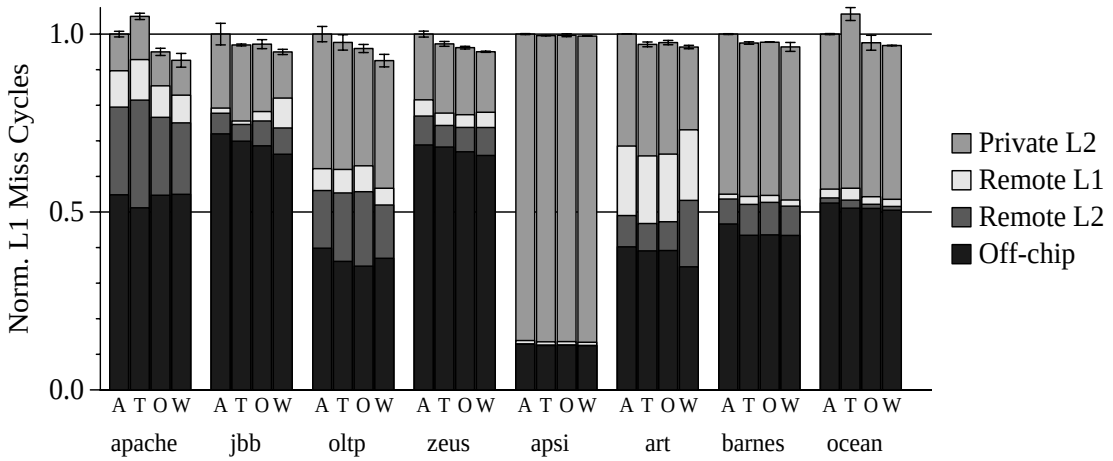


FIGURE 7-13. Private CMP-TLC *w/ASR*: L1 Miss Cycles Breakdown (A: ASR, T: Private CMP-TLC w/ASR, O: Private CMP-TLC-Request w/ASR, W: Private CMP-TLC-WDM w/ASR)

7.3.4 Evaluation: Interaction with ASR

Combining ASR with Private CMP-TLC requires some slight modifications to ASR. To account for the lower remote cache hit latency provided by transmission lines, ASR reduces the local hit benefit in its monitoring functions. Specifically, ASR with Private CMP-TLC and Private CMP-TLC-Request reduce the local hit benefit by 4 cycles and ASR with Private CMP-TLC-WDM reduces the local hit benefit by 10 cycles. Similarly, the off-chip miss cost is reduced by 10 cycles for Private CMP-TLC and Private CMP-TLC-Request and 15 cycles for Private CMP-TLC-WDM.

Since ASR utilizes remote on-chip caches more than the previous baseline CMP-Private design, transmission lines improve ASR's performance by a larger percentage. Figure 7-13 plots the normalized L1 miss cycles and displays transmission lines reduce L1 miss cycles by 0-10% for all workloads except for Private CMP-TLC running Apache. For Apache, ASR adapts replication to allow only a few replicas to exist on-chip, thus increasing remote cache utilization and transmission line bandwidth contention. The result is Private CMP-TLC exhibits 32% slower remote L2 hit latency than the baseline design, causing a significant performance degradation.

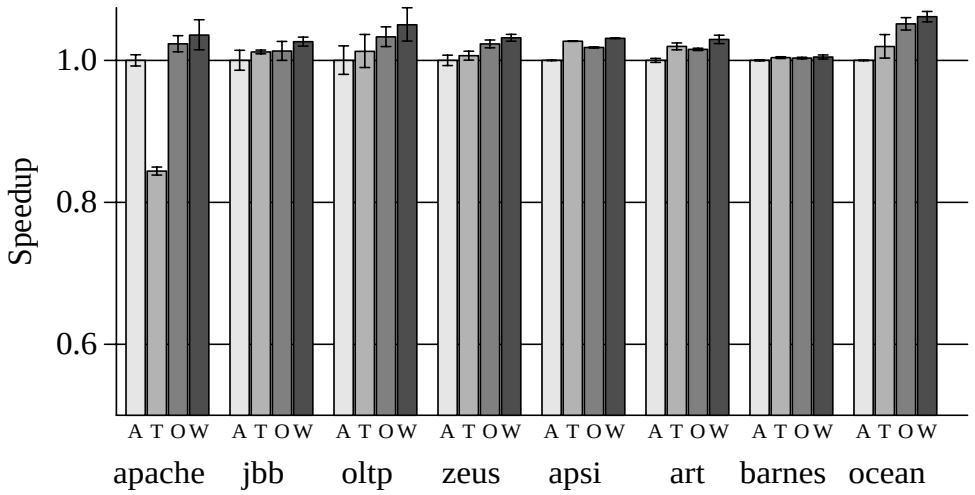


FIGURE 7-14. Private CMP-TLC *w/ASR*: Speedup
(A: ASR, T: Private CMP-TLC w/ASR, O: Private CMP-TLC-Request w/ASR
W: Private CMP-TLC-WDM w/ASR)

Figure 7-14 relates the performance of the Private CMP-TLC designs to that of ASR using conventional wires and shows that transmission lines can provide up to an additional 6% performance improvement to ASR. Similar to the previously evaluated transmission line designs, the Private CMP-TLC with ASR designs with higher bandwidth perform better. Specifically, the average performance improvement provided by Private CMP-TLC, Private CMP-TLC-Request, and Private CMP-TLC-WDM is 0%, 2%, 4%, respectively. In comparison, the average performance improvement provided by Private CMP-TLC, Private CMP-TLC-Request, and Private CMP-TLC-WDM without ASR was 1%, 1%, 3%. Therefore, as long as the transmission line network provides sufficient bandwidth, the combination of ASR and transmission lines performs better than either technique alone.

7.3.5 Sharing Type Latency vs. Off-chip Misses

This subsection shows more precisely why the combination of ASR and transmission lines performs better than either isolated technique. Specifically, the subsection provides further detail about the interaction between the ASR and transmission line by correlating on-chip latency with off-chip misses. Figure 7-15 illustrates the L1 miss latency to on-chip single requestor, shared read-only, and shared read-write blocks,

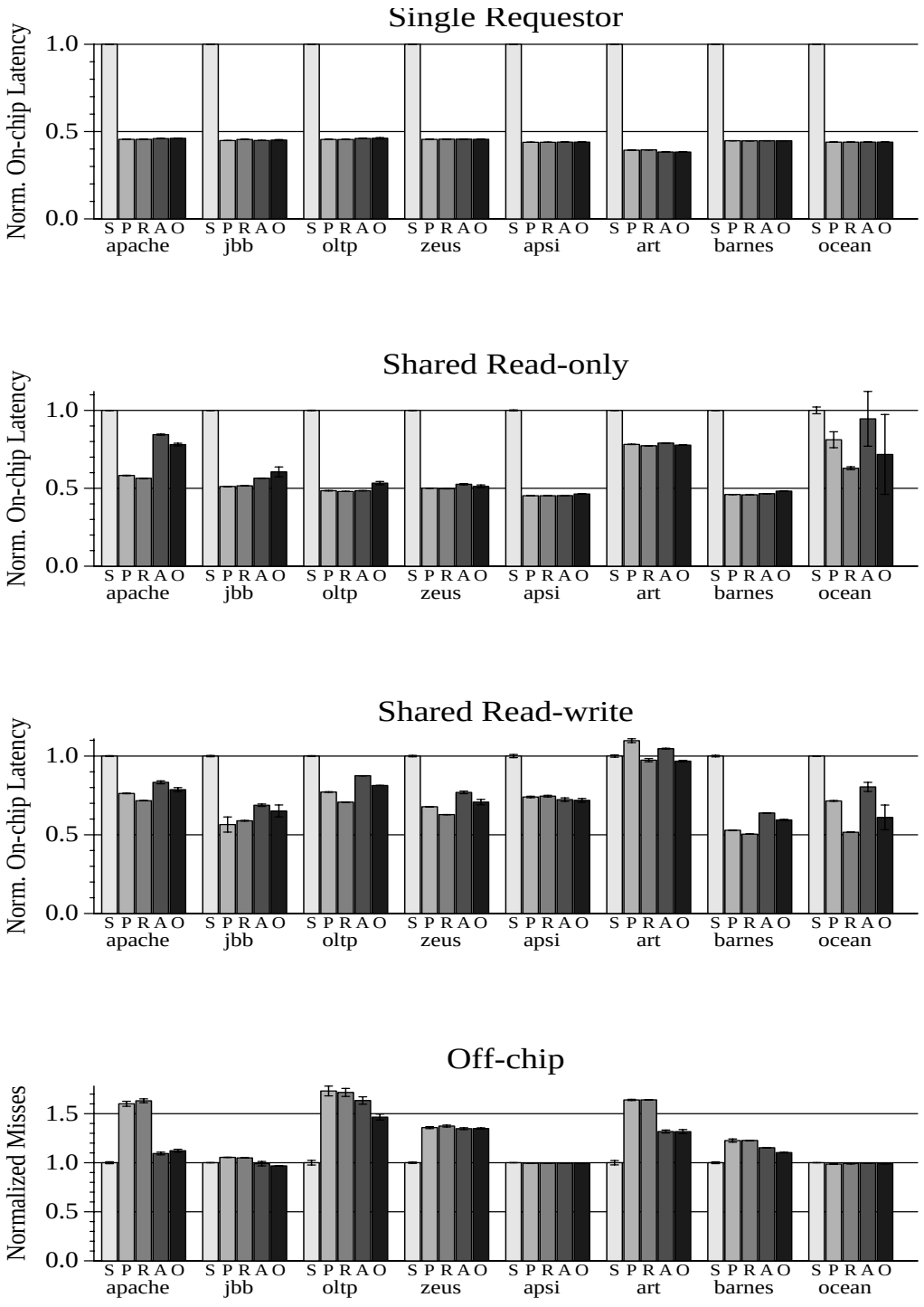


FIGURE 7-15. Normalized L1 Miss Latency to Sharing Types and Off-chip Misses (S: CMP-Shared, P: CMP-Private, R: Private CMP-TLC-Request, A: ASR, O: Private CMP-TLC-Request w/ ASR)

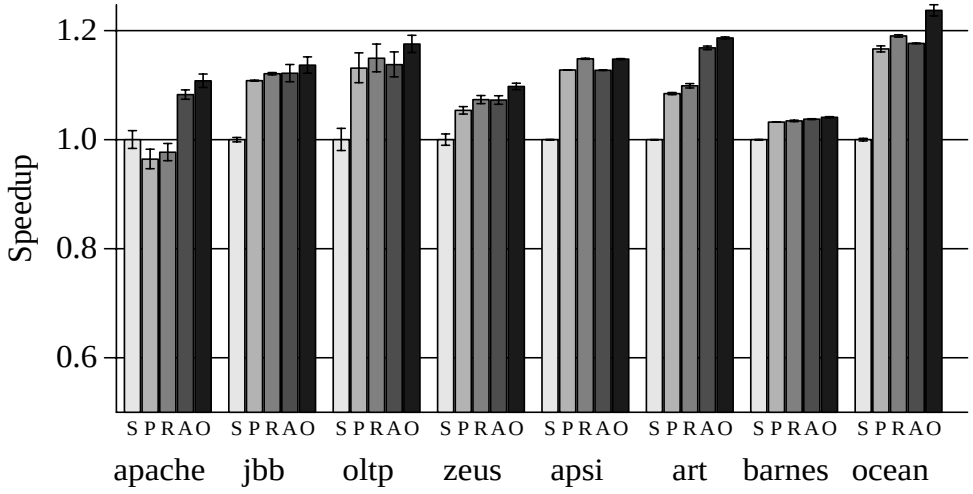


FIGURE 7-16. Combination of Techniques: Speedup (S: CMP-Shared, P: CMP-Private, R: Private CMP-TLC-Request, A: ASR, O: Private CMP-TLC-Request w/ ASR)

as well as, off-chip misses. Since virtually all single requestor hits are to a processor's local L2 cache, single requestor latency is unaffected by ASR or transmission lines. However, shared block latency increases or decreases depending on how Private CMP-TLC-Request with ASR (bars labeled *) chooses to exploit the lower latency of transmission lines. In particular, for the workloads Jbb, Oltp, Barnes and Ocean, ASR decreases replication, resulting in a 2-25% increase in shared read-only latency versus ASR without transmission lines (bars labeled A). The benefit of this decrease in replication is that off-chip misses reduce by 1-17% for these four workloads. On the other hand, ASR uses transmission lines to reduce the shared read-only latency for Apache, Zeus, Art, and Ocean, as well as, reduce shared read-write latency for all workloads. Overall, Figure 7-16 displays the combination of ASR and transmission lines always improves performance versus either technique alone.

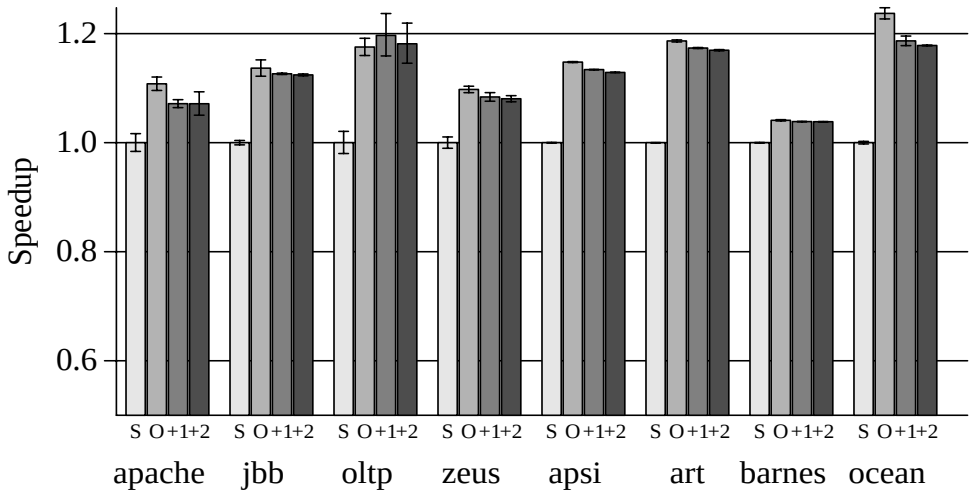


FIGURE 7-17. Private CMP-TLC-Request with ASR *Transceiver Sensitivity*: Speedup (S: CMP-Shared, O: Private CMP-TLC-Request with ASR, +1: Shared CMP-TLC with one extra transceiver delay cycle, +2: Shared CMP-TLC with two extra transceiver delay cycles)

Transceiver Delay Sensitivity Analysis. Similar to the previous Shared CMP-TLC analysis, this subsection evaluates the sensitivity of Private CMP-TLC-Request with ASR to an extra one and two cycles of transceiver delay. The sensitivity analysis focuses on Private CMP-TLC-Request with ASR because it was the best performing Private CMP-TLC design that didn't rely upon wave division multiplexing. For Private CMP-TLC-Request with ASR, the local hit benefit and off-chip miss cost are adjusted to match the longer latencies associated with the slower transceivers. Figure 7-17 plots how transceiver delay affects the performance of Private CMP-TLC-Request with ASR, and similar to Shared CMP-TLC, the extra transceiver delay causes consistent performance loss for all workloads except Barnes. However, the performance degradation of Private CMP-TLC with ASR is less because fewer requests communicate across the transmission lines. Specifically, while two extra transceiver delay cycles resulted in an average 3% performance degradation for Shared CMP-TLC, the average performance degradation for Private CMP-TLC-Request with ASR is 2%.

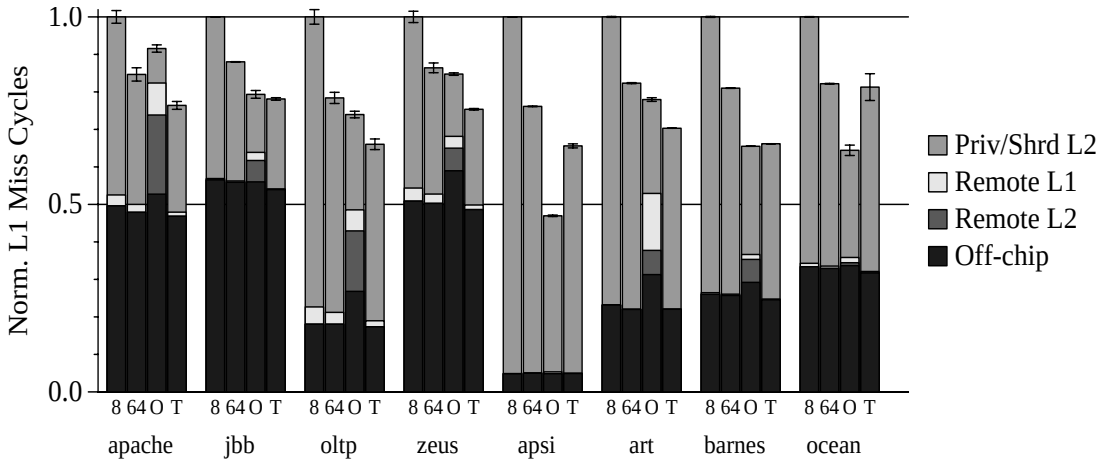


FIGURE 7-18. Best Performing Comparison: L1 Miss Cycles Breakdown (8: CMP-Shared, 64: CMP-SNUCA, O: Private CMP-TLC-Request w/ASR, T: Shared CMP-TLC)

7.4 Comparing Best Performing Designs

Though Private CMP-TLC-Request with ASR combines all three wire management techniques, Shared CMP-TLC achieves better performance for half the workloads because it uses smaller and faster L2 banks. This section compares the best two designs that don't require wide transmission line links: Private CMP-TLC-Request with ASR and Shared CMP-TLC, with the two conventional shared CMP designs: CMP-Shared and CMP-SNUCA. The difference between the two shared cache designs is CMP-Shared use 8 banks each with a 20-cycle access time, and CMP-SNUCA uses 64 banks each with 9-cycle access time [1]. First, Figure 7-18 compares the designs' L1 miss cycles. For the four workloads with the least amount of remote cache hit cycles: Jbb, Apsi, Barnes, and Ocean, Private CMP-TLC-Request with ASR achieves equivalent if not lower L1 misses cycles than Shared CMP-TLC. However, for the four remaining workloads that stress remote cache access latency: Apache, Oltp, Zeus, and Art, Shared CMP-TLC significantly reduces the L1 miss cycles versus the other three designs. Most of Shared CMP-TLC's cycle reduction is attributed to the latency benefit provided by its 64-banked L2 cache. By comparing the 'Shared L2' bars of

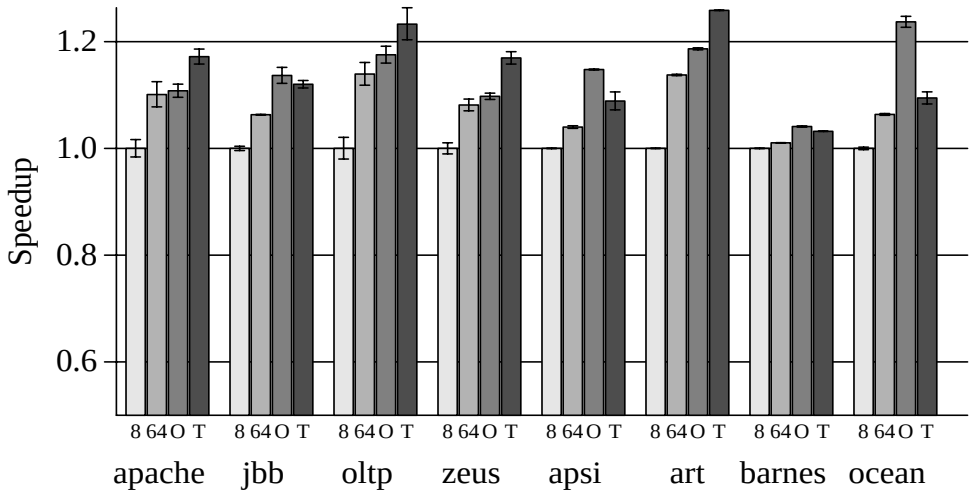


FIGURE 7-19. Best Performing Comparison: Speedup (8: CMP-Shared, 64: CMP-SNUCA, O: Private CMP-TLC-Request w/ASR T: Shared CMP-TLC)

CMP-Shared and CMP-SNUCA, one can observe the latency benefit of the 64-banked L2 cache. In particular, for hits in its shared L2 cache, CMP-SNUCA encounters approximately 27% less cycles than CMP-Shared.

Overall, both designs achieve equivalent average performance, but Private CMP-TLC-Request with ASR requires more storage overhead, while Shared CMP-TLC demands more upper layer metal tracks. Figure 7-19 shows both Private CMP-TLC-Request with ASR and Shared CMP-TLC improves performance by 14%, on average, versus CMP-Shared. To attain this performance, Private CMP-TLC-Request with ASR needs 211 KB of storage to support ASR's dynamic monitoring and 8 4-byte wide transmission line links traveling a maximum distance of 9 mm. In contrast, Shared CMP-TLC' rather simple architecture needs no extra storage, but demands 16 8-byte wide transmission line links between 9-13 mm. One could imagine Private CMP-TLC-Request's thinner transmission line network could be integrated into existing upper metal layers, while Shared CMP-TLC's wider transmission line network probably necessitates an additional metal layer. In the end, the CMP designer must decide whether the performance provided by on-chip transmission lines and ASR justify their respective cost in upper-layer wire tracks and substrate area.

7.5 Related Work

Other researchers have also studied the architectural impact of fast on-chip global communication. For instance, Balasubramonian *et al.* [10] and Cheng *et al.* [24] recently applied heterogeneous interconnect to global communication within a clustered processor and CMP, respectively. Heterogeneous interconnection networks are composed of multiple sets of physical wires, with each set implemented to a different point in the energy/delay/bandwidth spectrum. Both of these studies optimize global messages to utilize one of four networks—delay optimal, bandwidth optimal, power optimal, and power-bandwidth optimal—depending on the message's characteristics. In contrast, the cache designs discussed in this chapter use only conventional wires and transmission lines and thus don't require muxing four different physical channels.

Also Peh and Dally [85] proposed using a fast on-chip control message network to configure a slower, high-bandwidth data network to improve throughput. By scheduling switches ahead of data arrival, enabled buffers to only be held during actual buffer usage and reduced routing and arbitration latency. While utilizing fast on-chip wires, their flit-reservation flow control technique deals with low-level interconnect design issues, while the evaluated TLC designs deal with high-level interactions between coherence protocol and the wire technology.

7.6 Summary

On-chip transmission lines offer a significant latency advantage to conventional interconnect for long distance communication. One attractive application of transmission lines is utilizing them to access large CMP caches. This chapter showed transmission lines supply the best performance improvement to a shared CMP cache, but transmission lines can also improve private CMP cache performance. The chapter demonstrated transmission lines can work in harmony with other latency management techniques, such as Adaptive Selective Replication, to improve performance beyond what can be achieved by either technique alone. Finally, the chapter illustrated the Private CMP-TLC design that combined all three latency management techniques performs competitively with the Shared CMP-TLC design despite using larger and slower cache banks.

Chapter 8

Conclusions and Future Work

This chapter provides the dissertation's conclusions and discusses some possible areas of future work.

8.1 Conclusions

The on-chip cache hierarchy significantly impacts overall CMP performance. The conflicting requirements of reducing off-chip misses and managing slow global on-chip wires makes CMP cache design particularly challenging. Currently, CMPs utilize either a shared cache to reduce off-chip misses, or private caches to minimize on-chip latency. Ideally, architects desire a hybrid CMP cache that achieves both, but the tradeoff between the reducing off-chip misses and minimizing on-chip latency depends on workload behavior and system characteristics.

In order to improve CMP cache performance, this dissertation first characterized multithreaded workloads and then examined three techniques that exploited different workload behavior. In particular, this dissertation evaluated both commercial and scientific workloads and identified three sharing types that existed in these workloads: single requestor, shared read-only, and shared read-write. The characterization showed that improving scientific workload performance mostly depended on improving single requestor latency, while improving commercial workload performance more depended on improving shared read-only and shared read-write latency. Furthermore, for the commercial workloads, the characterization showed shared read-only data exhibited a very high request locality in and shared read-write data displayed migratory sharing behavior.

The first technique the dissertation investigated was migrating cache blocks within a shared CMP cache. While previous results showed block migration reduced wire delay in uniprocessor caches, this dissertation discovered block migration's capability to improve CMP performance relies on a difficult to implement smart search mechanism. Furthermore, the large amount of inter-processor sharing that exists in some workloads fundamentally limits block migration's benefit.

The second technique the dissertation proposed was Adaptive Selective Replication, which dynamically adapted 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 adjusted the degree of replication to match the current workload behavior and system configuration. The dissertation demonstrated ASR improves performance versus other hybrid designs and provides performance stability by always performing at least comparatively to the best alternative design.

The third technique the dissertation presented was using transmission lines to reduce on-chip latency in both a shared and private CMP cache. These Transmission Line Caches consistently improved performance, but bandwidth contention prevent some TLCs from achieving their full potential. By isolating transmission lines to only low-bandwidth latency-critical messages, thin transmission line links where able to achieve most of the performance improvement provided by much wider transmission line links. However, if the transmission line links were over-subscribed communicating wide messages, their latency advantage was partially, if not fully, negated by bandwidth contention.

8.2 Future Work

For the ideas proposed in the dissertation, this section outlines potential directions of future work.

Energy Evaluation. This dissertation did not evaluate the energy impact migration, selective replication, and transmission lines will have in a CMP cache. However, other results have shown cache block migration increases CMP energy consumption [15, 48]. Selective replication could increase or decrease energy

consumption depending on the energy consumption ring writebacks have compared to off-chip communication. One would suspect ring writebacks require less energy than utilizing the power-hungary off-chip drivers, but such a suspicion necessitates further investigation. While Chapter 7 showed, using first-order equations, that on-chip transmission lines demand less energy than conventional wires, this dissertation did not compare the TLCs' overall energy demand to conventional cache designs.

Bloom Filters for ASR. Replacing ASR's partial tag structures with bloom filters, could reduce its storage overhead. Chapter 6 demonstrated that ASR's Next-Level Hit Buffer and Victim Tag Buffer accounted for the majority of its storage overhead. Both buffers stored the low-order bits of cache tags to estimate the benefit of increasing or decreasing replication. Instead, the buffers could be implemented as non-counting bloom filters and use false positive feedback to determine when the filters should be reset. Such bloom filters have been shown to significantly reduce storage overhead versus partial tags in other similar applications [92]. However, an open question for ASR bloom filters is determining false positives.

ASR for Lower Power. The current ASR proposal adjusts replication to achieve lower memory cycles, however, ASR could adapt replication for lower power. While it is unclear if ASR's power savings would justify its costs, the changes required for a "power aware" ASR would be straightforward. One would simply change ASR's local hit cycle benefit and off-chip miss cycle cost constants to local hit power benefit and off-chip miss power cost constants, respectively.

Multithreaded Cores. The CMP cores evaluated in this dissertation are all single-threaded. However, future CMP cores are likely to be multithreaded. Multithreaded cores would likely place additional bandwidth demands on a CMP cache and could potentially tolerate slower L2 caches. Because the techniques evaluated in this dissertation are sensitive to bandwidth demand and L2 latency tolerance, future research should explore multithreaded cores.

References

[1] Vikas Agarwal, Stephen W. Keckler, and Doug 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] Ardsher Ahmed, Pat Conway, Bill Hughes, and Fred Weber. AMD Opteron Shared Memory MP Systems. In *Proceedings of the 14th HotChips Symposium*, August 2002.

[3] Alaa R. Alameldeen, Milo M. K. Martin, Carl J. Mauer, Kevin E. Moore, Min Xu, Daniel J. Sorin, Mark D. Hill, and David A. Wood. Simulating a $2M Commercial Server on a $2K PC. *IEEE Computer*, 36(2):50–57, February 2003.

[4] Bharadwaj S. Amrutur and Mark A. Horowitz. Speed and Power Scaling of SRAMs. *IEEE Transactions on Solid-State Circuits*, 35(2):175–185, February 2000.

[5] Tom M. Apostol. *Calculus*. Ginn (Blaisdell), 1969.

[6] Vishal Aslot, Max Domeika, Rudolf Eigenmann, Greg Gaertner, Wesley Jones, and Bodo Parady. SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance. In *Workshop on OpenMP Applications and Tools*, pages 1–10, July 2001.

[7] Avanti. *Star-Hspice Manual*, Jan 1999.

[8] Peng Bai and et al. A 65nm Logic Technology Featuring 35 nm Gate Length, Enhanced Channel Strain, 8 Cu Interconnect Layers, Low-k ILD and 0.57 m SRAM Cell*Electron Devices Meeting, IEDM Technical Digest. International*, December 2004.

[9] J. Balachandran, S. Brebels, G. Carchon, T. Webers, W. De Raedt, B. Nauwelaers, and E. Beyne. Package Level Interconnect Options. In *International Workshop on System level Interconnect Prediction*, pages 21–27, April 2005.

[10] Rajeev Balasubramonian, Naveen Muralimanohar, Karthik Ramani, and Venkatanand Venkatachalapathy. Microarchitectural Wire Management for Performance and Power in Partitioned Architectures. In *Proceedings of the 11th IEEE Symposium on High-Performance Computer Architecture*, February 2005.

[11] Paul Barford and Mark Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Proceedings of the 1998 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 151–160, June 1998.

[12] Luiz A. Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory System Characterization of Commercial Workloads. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 3–14, June 1998.

[13] Bradford M. Beckmann, Michael R. Marty, and David A. Wood. ASR: Adaptive Selective Replication for CMP Caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2006.

[14] Bradford M. Beckmann and David A. Wood. TLC: Transmission Line Caches. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2003.

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

[16] Bradley J. Benschneider and et al. A 300-MHz 64-b Quad-Issue CMOS RISC Microprocessor. *IEEE Journal of Solid-State Circuits*, 30(11):1203–1214, Nov 1995.

[17] A. S. Brown. Fast Films. *IEEE Spectrum*, 20(2):36–40, February 2003.

[18] Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture. In *Proceedings of the 11th IEEE Symposium on High-Performance Computer Architecture*, February 2005.

[19] Rohit Chandra, Scott Devine, Ben Verghese, Anoop Gupta, and Mendel 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)*, October 1994.

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

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

[22] Alan Charlesworth. Starfire: Extending the SMP Envelope. *IEEE Micro*, 18(1):39–49, Jan/Feb 1998.

[23] Guoqing Chen, Hui Chen, Mikhail Haurylau, Nicholas Nelson, Philippe M. Fauchet, and Eby G. Friedman. Predictions of CMOS Compatible On-Chip Optical Interconnect. In *International Workshop on System level Interconnect Prediction*, pages 13–20, April 2005.

[24] Liqun Cheng, Naveen Muralimanohar, Karthik Ramani, Rajeev Balasubramonian, and John B. Carter. Interconnect-Aware Coherence Protocols for Chip Multiprocessors. In *Proceedings of the 33nd Annual International Symposium on Computer Architecture*, June 2006.

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

[26] Zeshan Chishti, Michael 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.

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

[28] William J. Dally and John W. Poulton. *Digital Systems Engineering*. Cambridge University Press, 1998.

[29] Evan E. Davidson. Large chip vs. MCM for a High-Performance System. *IEEE Micro*, 18(4):33–41, July/Aug 1998.

[30] Evan E. Davidson, Bradley D. McCredie, and Walter V. Vilkelis. Long Lossy Lines (L/sup 3/) and Their Impact Upon Large Chip Performance. *IEEE Transactions on Components, Packaging and Manufacturing Technology, Part B: Advanced Packaging*, 20(4):361–375, November 1997.

[31] Alina Deutsch. Electrical Characteristics of Interconnections for High-Performance Systems. *Proceedings of the IEEE*, 86(2):315–355, February 1998.

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

[33] Antonije R. Djordjevic, Miodrag B. Bazdar, Tapan K. Sarkar, and Roger F. Harrington. *Matrix Parameters for Multiconductor Transmission Lines: Software and User's Manual*. Artech House, 1989.

[34] Susan J. Eggers and Randy H. Katz. A Characterization of Sharing in Parallel Programs and its Application to Coherency Protocol Evaluation. In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pages 373–382, May 1988.

[35] Babak Falsafi and David 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.

[36] Pierfrancesco Foglia, Daniele Mangano, and Cosimo Antonio Prete. A NUCA Model for Embedded Systems Cache Design. In *3rd IEEE 2005 Workshop on Embedded Systems for Real-Time Multimedia (ESTIMEDIA)*, September 2005.

[37] International Technology Roadmap for Semiconductors. ITRS 2003 Edition. Semiconductor Industry Association, 2003. http://public.itrs.net/Files/2003ITRS/Home2003.htm.

[38] International Technology Roadmap for Semiconductors. ITRS 2004 Update. Semiconductor Industry Association, 2004. http://www.itrs.net/Common/2004Update/2004Update.htm.

[39] International Technology Roadmap for Semiconductors. ITRS 2005 Edition. Semiconductor Industry Association, 2005. http://www.itrs.net/Common/2005ITRS/Home2005.htm.

[40] Steven Frank, Henry Burkhardt, III, and James Rothnie. The KSR1: Bridging the Gap Between Shared Memory and MPPs. In *Proceedings of the 38th Annual IEEE Computer Society Computer Conference (COMPCON)*, pages 285–295, February 1993.

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

[42] Anoop Gupta and Wolf-Dietrich Weber. Cache Invalidation Patterns in Shared-Memory Multiprocessors. *IEEE Transactions on Computers*, 41(7):794–810, July 1992.

[43] Erik Hagersten, Anders Landin, and Seif Haridi. DDM–A Cache-Only Memory Architecture. *IEEE Computer*, 25(9):44–54, September 1992.

[44] Sarah L. Harris. *Synergistic Caching in Single-chip Multiprocessors*. PhD thesis, Department of Electrical Engineering, Stanford University, 2005.

[45] Mark D. Hill and Alan Jay Smith. Evaluating Associativity in CPU Caches. *IEEE Transactions on Computers*, 38(12):1612–1630, December 1989.

[46] Ron Ho, Kenneth W. Mai, and Mark A. Horowitz. The Future of Wires. *Proceedings of the IEEE*, 89(4):490–504, April 2001.

[47] M. S. Hrishikesh, Norman P. Jouppi, Keith I. Farkas, Doug Burger, Stephen W. Keckler, and Premkishore Shivakumar. The Optimal Logic Depth Per Pipeline Stage is 6 to 8 Inverter Delays. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, May 2002.

[48] Jaehyuk Huh, Changkyu Kim, Hazim Shafi, Lixin Zhang, Doug Burger, and Stephen W. Keckler. A NUCA Substrate for Flexible CMP Cache Sharing. In *Proceedings of the 19th International Conference on Supercomputing*, June 2005.

[49] Nathan Ida. *Engineering Electromagnetics*. Springer, 2000.

[50] Aamer Jaleel, Matthew Mattina, and Bruce 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*, February 2006.

[51] Dale W. Jorgenson, Mun S. Ho, and Kevin J. Stiroh. *Information Technology and the American Growth Resurgence*. MIT Press, 2005.

[52] Anup P. Jose, George Pataunakis, and K. L. Shepard. Near speed-of-light on-chip interconnects using pulsed current-mode signalling. In *Proceedings of the 2005 Symposium on VLSI Circuits*, pages 108–111, 2005.

[53] Ron Kalla, Balaram Sinharoy, and Joel M. Tendler. IBM Power5 Chip: A Dual Core Multi-threaded Processor. *IEEE Micro*, 24(2):40–47, Mar/Apr 2004.

[54] Pawan Kapur and Krishna C. Saraswat. Comparison Between Electrical and Optical Interconnects for On-chip Signaling. pages 89–91, 2002.

[55] Chetana N. Keltcher, Kevin J. McGrath, Ardsher Ahmed, and Pat Conway. The AMD Opteron Processor for Multiprocessor Servers. *IEEE Micro*, 23(2):66–76, March-April 2003.

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

[57] Sunil P. Khatri and et al. A Novel VLSI Layout Fabric for Deep Sub-Micron Applications. In *Design Automation Conference*, pages 491–496, June 1999.

[58] Changkyu Kim, Doug Burger, and Stephen 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)*, October 2002.

[59] Seongbeom Kim, Dhruba Chandra, and Yan Solihin. Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture. In *Proceedings of the 2004 International Conference on Parallel Architectures and Compilation Techniques*, pages 111–222, 2004.

[60] Mauro J. Kobrinsky, Bruce A. Block, Jun-Fei Zheng, Brandon C. Barnett, Edris Mohammed, Miriam Reshotko, Frank Robertson, Scott List, Ian Young, and Kenneth Cadien. On-Chip Optical Interconnects. *Intel Technology Journal*, May 2004.

[61] Akio Kodama and Toshinori Sato. A Non-Uniform Cache Architecture on Networks-on-Chip: A Fully Associative Approach with Pre-Promotion. In *10th International Symposium on Integrated Circuits, Devices and Systems*, September 2004.

[62] Poonacha Kongetira. A 32-way Multithreaded SPARCÆ Processor. In *Proceedings of the 16th HotChips Symposium*, August 2004.

[63] Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun. Niagara: A 32-Way Multi-threaded Sparc Processor. *IEEE Micro*, 25(2):21–29, Mar/Apr 2005.

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

[65] Kevin Krewell. UltraSPARC IV Mirrors Predecessor. *Microprocessor Report*, pages 1–3, November 2003.

[66] Rakesh Kumar, Victor Zyuban, and Dean Tullsen. Interconnections in multi-core architectures: Understanding Mechanisms, Overheads and Scaling. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, June 2005.

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

[68] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance*. Prentice Hall, 1984.

[69] Feihui Li, Chrysostomos Nicopoulos, Thomas Richardson, Yuan Xie, Vijaykrishnan Narayanan, and Mahmut Kandemir. Design and Management of 3D Chip Multiprocessors Using Network-in-Memory. In *Proceedings of the 33nd Annual International Symposium on Computer Architecture*, June 2006.

[70] Chun Liu, Anand Sivasubramaniam, and Mahmut 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*, February 2004.

[71] LostCircuits. Intel Pentium4 600 Series. http://www.lostcircuits.com/cpu/p4-600/, Feb 21, 2005.

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

[73] Atul Maheshwari and Wayne Burleson. Current Sensing Techniques for Global Interconnects in Very Deep Submicron (VDSM) CMOS. In *Proceedings of the IEEE 2001 Computer Society Workshop on VLSI*, pages 66–70, April 2001.

[74] Milo M. K. Martin. *Token Coherence*. PhD thesis, University of Wisconsin, 2003.

[75] Milo M.K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *Computer Architecture News*, pages 92–99, September 2005.

[76] Michael R. Marty, Jesse D. Bingham, Mark D. Hill, Alan J. Hu, Milo M. K. Martin, and David A. Wood. Improving Multiple-CMP Systems Using Token Coherence. In *Proceedings of the Eleventh IEEE Symposium on High-Performance Computer Architecture*, February 2005.

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

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

[79] David A. B. Miller. Ratioinale and Challenges for Optical Interconnects to Electronic Chips. *Proceedings of the IEEE*, 88(6):728–749, June 2000.

[80] Masayuki Minzuno, Kenichiro Anjo, Yoshikazu Sumi, Muneo Fukaishi, Hitoshi Wakabayashi, Tohru Mogami, Tadahiko Horiuchi, and Masakazu Yamashina. Clock Distribution Networks with On-Chip Transmission Lines. In *Proceedings of the IEEE 2000 International Interconnect Technology Conference*, pages 3–5, 2000.

[81] Haim E. Mizrahi, Jean-Loup Baer, Edward D. Lazowska, and John Zahorjan. Introducing Memory into the Switch Elements of Multiprocessor Interconnection Networks. In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, May 1989.

[82] Gordon E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, pages 114–117, April 1965.

[83] Shin'ichiro 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.

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

[85] Li-Shiuan Peh and William J. Dally. Flit-Reservation Flow Control. In *Proceedings of the Sixth IEEE Symposium on High-Performance Computer Architecture*, January 2000.

[86] Christopher Poirer, Richard McGowen, Christopher Bostak, and Samuel Naffziger. Power and temperature control on a 90nm Itanium-family processor. In *Proceedings of the IEEE 2005 International Solid-State Circuits Conference*, pages 304–305, 2005.

[87] Donald A. Priore. Inductance on Silicon for Sub-micron CMOS VLSI. In *Proceedings of the 1993 Symposium on VLSI Circuits*, pages 17–18, 1993.

[88] M. Racanelli and et al. Ultra High Speed SiGe NPN for Advanced BiCMOS Technology. *Electron Devices Meeting, IEDM Technical Digest. International*, pages 15.3.1–15.3.4, 2001.

[89] Alex Ramirez, Oliverio J. Santana, Josep L. Larriba-Pey, and Mateo Valero. Fetching instruction streams. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 371–382, November 2002.

[90] Sumant Ramprasad, Naresh R. Shanbhag, and Ibrahim N. Hajj. A Coding Framework for Low-Power Address and Data Busses. *IEEE Transactions on VLSI Systems*, 7(2):212–2, June 1999.

[91] Intel Press Release. Intel Desktop Processors Get 64-Bit Support. http://www.intel.com/press-room/archive/releases/20050221comp.htm, Feb 21, 2005.

[92] Robert Ricci, Steve Barrus, Dan Gebhardt, and Rajeev Balasubramonian. Managing Complexity in the Piranha Server-Class Processor Design. In *7th Workshop on Complexity-Effective Design held in conjunction with the 33rd International Symposium on Computer Architecture*, June 2006.

[93] Anand Lal Shimpi and Derek Wilson. Intel Pentium4 600 Series. http://www.anandtech.com/printarticle.aspx?i=2353, Feb 21, 2005.

[94] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. SPLASH: Stanford Parallel Applications for Shared Memory. *Computer Architecture News*, 20(1):5–44, March 1992.

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

[96] Vassos Soteriou and Li-Shiuan Peh. Design Space Exploration of Power-Aware On/Off Interconnection Networks. In *Proceedings of Internationl Conference on Computer Design (ICCD'04)*, October 2004.

[97] Vijayaraghavan Soundararajan, Mark Heinrich, Ben Verghese, Kourosh Gharachorloo, Anoop Gupta, and John 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.

[98] Evan Speight, Hazim Shafi, Lixin Zhang, and Ram 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.

[99] Per Stenström, Mats Brorsson, and Lars Sandberg. Adaptive Cache Coherence Protocol Optimized for Migratory Sharing. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 109–118, May 1993.

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

[101] G. Edward Suh, Srinivas Devadas, and Larry Rudolph. A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning. In *Proceedings of the Eighth IEEE Symposium on High-Performance Computer Architecture*, February 2002.

[102] Herb Sutter. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal*, 30(3), March 2005.

[103] Dennis Sylvester, William Jiang, and Kurt Keutzer. BACPAC - Berkeley Advanced Chip Performance Calculator website. http://www-device.eecs.berkeley.edu/ dennis/bacpac/.

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

[105] Joel M. Tendler, Steve Dodson, Steve Fields, Hung Le, and Balaram Sinharoy. POWER4 System Microarchitecture. IBM Server Group Whitepaper, October 2001.

[106] Joel M. Tendler, Steve Dodson, Steve Fields, Hung Le, and Balaram Sinharoy. POWER4 System Microarchitecture. *IBM Journal of Research and Development*, 46(1), 2002.

[107] Frank F. Tsui. JSP - A Research Signal Processor in Josephson Technology. *IBM Journal of Research and Development*, 24(2):243–252, March 1980.

[108] Ben Verghese, Scott Devine, Anoop Gupta, and Mendel 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)*, October 1996.

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

[110] J. D. Warnock and et al. The Circuit and Physical Design of the POWER4 Microprocessor. *IBM Journal of Research and Development*, 46(1):27–51, January 2002.

[111] Fred Weber. AMD's Next Generation Microprocessor Architecture, October 2001.

[112] N. Weste and K. Eshragian. *Principles of CMOS VLSI Design: A Systems Perspective*. Addison-Wesley Publishing Co., 1982.

[113] Steven J.E. Wilton and Norman P. Jouppi. An enhanced access and cycle time model for on-chip caches. *Tech report 93/5, DEC Western Research Lab*, 1994.

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

[115] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop 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.

[116] Chung-Yu Wu and Ming-Chuen Shiau. Delay Models and Speed Improvement Techniques for RC Tree Interconnections Among Small-Geometry CMOS Inverters. *IEEE Journal of Solid-State Circuits*, 25(5):1247–1256, Oct 1990.

[117] Thucydides Xanthopoulos, Daniel W. Bailey, Michael K. Gowan Atul K. Gangwar, Anil K. Jain, and Brian K. Prewitt. The Design and Analysis of the Clock Distribution Network for a 1.2 GHz

Alpha Microprocessor. In *Proceedings of the IEEE 2001 International Solid-State Circuits Conference*, pages 402–403, 2001.

[118] Thomas Y. Yeh and Glenn Reinman. Fast and Fair: Data-stream Quality of Service. In *Proceedings of the 2005 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, September 2005.

[119] Hui Zhang, Varghese George, and Jan M. Rabaey. Low-Swing On-Chip Signaling Techniques: Effectiveness and Robustness. *IEEE Transactions on VLSI Systems*, 8(3):264–272, June 2000.

[120] Michael Zhang and Krste Asanovic. Miss Tags for Fine-Grain CAM-Tag Cache Resizing. In *Proceedings of the International Symposium on Low Power Electronics and Design*, August 2002.

[121] Michael Zhang and Krste 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.

[122] Zheng Zhang and Josep Torrellas. Reducing Remote Conflict Misses: NUMA with Remote Cache versus COMA. In *Proceedings of the Third IEEE Symposium on High-Performance Computer Architecture*, February 1997.