Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors

Evan Speight, Hazim Shafi, Lixin Zhang, and Ram Rajamony
Novel Systems Architecture Group
IBM Research, Austin, TX 78758
{speight,hshafi,zhangl,rajamony}@us.ibm.com

Abstract

With the ability to place large numbers of transistors on a single silicon chip, manufacturers have begun developing chip multiprocessors (CMPs) containing multiple processor cores, varying amounts of level 1 and level 2 caching, and on-chip directory structures for level 3 caches and memory. The level 3 cache may be used as a victim cache for both modified and clean lines evicted from on-chip level 2 caches. Efficient area and performance management of this cache hierarchy is paramount given the projected increase in access latency to off-chip memory.

This paper proposes simple architectural extensions and adaptive policies for managing the L2 and L3 cache hierarchy in a CMP system. In particular, we evaluate two mechanisms that improve cache effectiveness. First, we propose the use of a small history table to provide hints to the L2 caches as to which lines are resident in the L3 cache. We employ this table to eliminate some unnecessary clean write backs to the L3 cache, reducing pressure on the L3 cache and utilization of the on-chip bus. Second, we examine the performance benefits of allowing write backs from L2 caches to be placed in neighboring, on-chip L2 caches rather than forcing them to be absorbed by the L3 cache. This not only reduces the capacity pressure on the L3 cache but also makes subsequent accesses faster since L2-to-L2 cache transfers have typically lower latencies than accesses to a large L3 cache array. We evaluate the performance improvement of these two designs, and their combined effect, on four commercial workloads and observe a reduction in the overall execution time of up to 13%.

1. Introduction

Several modern chip multiprocessor (CMP) systems employ a victim cache approach [10] in the lower levels of the memory hierarchy. Using L3 cache memory as a victim cache enables better utilization of cache capacity. Furthermore, since inclusion need not be maintained, an L3 victim cache and the system memory can have distinct data paths. This can better accommodate the bandwidth requirements of larger numbers of on-die processors and increased levels of hardware multithreading.

Although victim caches traditionally store modified lines, writing back clean lines from the L2 to the L3 can help eliminate subsequent expensive memory accesses. In such a system, properly managing which cache lines are written back to the L3 cache can have a dramatic impact on the overall performance of the system. Judicious use of write backs can lower both on-chip resource utilization as well as resource usage at the L2 and L3 levels. This paper examines simple, adaptive mechanisms at the architecture level for managing write backs with the goal of improving performance.

Three ongoing developments in microprocessor design make this problem important. First, manufacturers are able to place very large numbers of transistors on a single silicon chip, giving rise to chip multiprocessors containing an increasing number of processing cores, L1 and L2 caches, on-chip directory structures for L3 caches, and memory controllers [9, 13]. Second, the advent of silicon carrier and multi-chip module technology enables L3 data arrays to be placed in close proximity to the processor chip with potentially dedicated access links. Third, in an effort to extract more thread-level parallelism from modern workloads, chips are designed with increasing numbers of hardware thread contexts per processor core. Disruptive interactions between the working sets of these threads can cause significant cache performance degradation due to L2 cache capacity misses.

Figure 1 shows the chip multiprocessor (CMP) architecture organization we examine in this paper. The CMP consists of eight CPU cores, each with two hardware SMT threads and private L1 Harvard-style instruction and data caches. Each pair of CPUs shares an L2 cache through the core interface unit (CIU). The CIU also routes requests to

0-7695-2270-X/05/$20.00 (C) 2005 IEEE

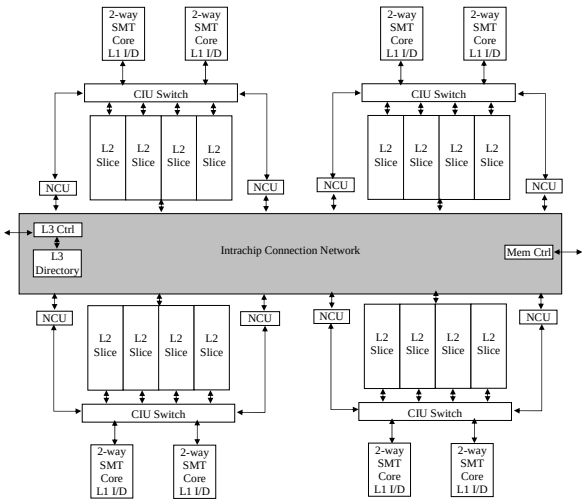

Figure 1. Baseline Architecture

and from each of the per-CPU non-cacheable units (NCU). Each L2 is divided into four slices to allow for simultaneous concurrent access to four regions of the cache. Physical memory addresses are statically mapped to one distinct slice in each L2 cache.

The L2 caches are the points of coherence in the system. They are connected to each other, the L3 cache, and memory controller through a point-to-point, bi-directional intrachip ring network. The coherence protocol implemented is an extension of that found in IBM's POWER 4 systems, which supports cache-to-cache transfers (interventions) for all dirty lines and a subset of lines in the shared state [13]. The L3 resides on its own dedicated off-chip pathway that is distinct from the pathway to and from memory. Both dirty and clean lines are written back from the L2 caches to the L3 cache in order to reduce the access time to these lines on a subsequent miss. A miss to a line that resides both in another on-chip L2 cache and the L3 cache will be serviced via an L2 cache-to-cache transfer, as this intervention is faster than retrieving the line from the off-chip L3 cache (See Section 4).

For chip multiprocessor systems such as the one described above, we propose two mechanisms for managing write backs:

- **Limit the number of unnecessary clean write backs from the L2 caches.** We first consider the case where write backs from the L2 caches can only go to the L3 cache. Dirty lines victimized in the L2 caches need to always be written back to the L3 cache. Such write backs are necessary since only one valid copy of the line exists in the system. *Clean* lines may also be chosen for replacement in L2 caches. The access latency to memory is already much higher than that to an L3 cache. This latency gap is projected to be substantially higher in future designs, and especially so when silicon carrier technologies could allow an L3 cache to be brought "on-chip". Writing back clean victimized lines to the L3 thus enables subsequent accesses to these lines to only pay the L3 access latency rather than the full memory access penalty.

  However, writing back clean victimized lines is unnecessary if another cache (L2 or L3) already has a valid copy of the line. An indiscriminate policy that writes back all clean victimized lines will consume valuable intrachip ring bandwidth as well as put pressure on other system resources. We regulate the write back of clean victimized lines by introducing a small table at each L2 cache that provides a hint as to whether a cache line is likely to be in the L3 or not. This table is consulted in order to make a more informed decision whether or not to write back a clean line to the L3 cache. Note that an incorrect decision only affects performance, not correctness.

- **Write back L2 cache lines to peer L2 caches.** Here, we consider the case where write backs from an L2 cache can be placed in a peer L2 cache. Because neighboring L2 caches are on-chip, providing data via an L2–to–L2 transfer is more than twice as fast when compared to retrieving the line from the L3 cache. As long as there is room available at a peer L2 cache, we could retain victimized lines (both clean and dirty) on–chip instead of placing them in the L3. However, on–chip L2 capacity is very precious compared to off–chip L3 capacity. When writing back a clean line to a peer L2 cache, we therefore need to be acutely aware of the reuse potential for that line.

  We use a second adaptive mechanism based again on a small table to track which lines have high reuse potential and should be kept on-chip if possible. This table is consulted when lines are written back from the L2 and enables lines with high reuse potential to be picked up by a peer L2 and those for which reuse information is unknown to be placed in the L3.

We examine the performance benefits of the two mechanisms using a set of commercial workload traces gathered from real machines that represent on-line transaction processing (TP, CPW2), enterprise Java computing (Trade2), and e-mail processing (NotesBench). We feed traces of these applications into a simulator with a detailed, cycle-accurate model of the architecture depicted in Figure 1. Results on these applications show improvements of up to 13% over the baseline system.

The rest of this paper is organized as follows. Sections 2 and 3 describe the architectural mechanisms and coherence

policies we propose to address the problems listed above. Section 4 provides details on our experimental setup and applications used. Section 5 gives performance results based on our detailed simulations. Section 6 talks briefly about related work, and Section 7 concludes the paper.

2. Selective Clean L2 Write Backs

When a dirty line must be evicted from an L2 cache to make room for an incoming line, the L2 cache has no choice: the dirty line must be sent out to somewhere else in the memory hierarchy. Therefore, for dirty lines, the decision on which lines to accept and which lines to reject is made by the L3 cache. Accepted lines are placed in the L3 victim cache. Rejected lines may either generate a retry bus response from the L3, or may be accepted by memory and incur a full memory latency if accessed later. We model the former protocol. Lines may be rejected by the L3 if there are not enough hardware resources to take the line immediately (e.g., the incoming data queue is full).

Although clean write backs potentially increase pressure on the intrachip ring and off-chip pins, subsequent L2 cache misses to these lines will likely be serviced by the faster L3 rather than memory. However, if the number of outstanding loads per thread increases, the performance improvements will diminish due to contention for the intrachip ring and shared L3 cache. Here we describe an adaptive mechanism to limit the number of clean lines written back from on-chip L2 caches to the off-chip L3 cache in an effort to reduce the negative impact of contention.

Indiscriminately evicting clean line from the L2 caches to the L3 cache means that clean lines may be written back even though they are already valid in the L3 cache. This can happen either because another peer L2 cache wrote the line back or the line has been replaced and missed on repeatedly by the same L2. This negatively impacts performance by increasing demands on both the CMP address and data rings, and by increasing the snoop processing required at all bus agents. Table 1 shows that for all our studied applications, the percentage of clean lines written back to the L3 cache that are already valid in the L3 can be greater than 50%.

CPW	60.0%
NotesBench	59.1%
TP	42.1%
Trade2	79.1%

Table 1. Percentage of Clean L2 Write Backs Already Present in the L3 Cache

Our simulated baseline coherence protocol will cancel the data ring transfer when an L2 clean write back request hits in the L3 cache, somewhat ameliorating the effect of these "unnecessary" clean write backs on the data bus. To further reduce pressure on the memory subsystem in the presence of high contention, the first component of our protocol is a *filter* that selectively decides which replaced clean lines should be sent from the L2 to the L3. If a wrong choice is made (e.g., the decision not to write back a clean line is made at the L2, but the line is actually not present in the L3 or another L2), an access to the line incurs a full memory latency if it is referenced later. The filtering step takes place at every L2 cache in the chip multiprocessor. We consider the case where the L2 filtering components operate independently of each other and the case where the filtering components co-operate so that only lines not present at any other L2 cache are sent to the L3 cache.

The proposed selective write back mechanism uses a small lookup table called the Write Back History Table (WBHT). This table is associated with each L2 cache in the system, and is organized and accessed just like a cache tag array. The following steps outline how the WBHT reduces unnecessary clean write backs:

1. When a clean line is written back from the L2 cache, the write back request is sent out on the intrachip ring.
2. The L3 cache snoop response indicates that it does or does not currently have the line in a valid state.
3. When the L2 caches receive the combined bus response, an entry for the line is allocated in the WBHT if the line is present in the L3 cache.
4. The next time this line is brought into the cache and subsequently chosen for replacement by whatever replacement policy is in force for the L2 cache, the table is checked to see if a corresponding entry exists. If so, the write back is deemed unnecessary and is aborted. Otherwise, the line is written back to the L3 cache.
5. Lines disappear from the WBHT due to the fact that there are many fewer entries than possible tag values, ensuring that lines that have not been accessed for a long time will lose their place in the table using an LRU policy.

Note that this table may be accessed after a line has been evicted from the cache and placed in the write back queue associated with the local L2 cache. Therefore, access to this table is not on the critical path of a cache miss, and we do not find it unreasonable to assume a fairly large table size (32K tag entries, or about 9% of our L2 cache size) and level of associativity (16-way).

2.1. Potential Negative Effects

Because the information in the WBHT serves only as a performance hint, there is no correctness issue if the con-

tents of the WBHT begin to diverge from the actual contents of the L3 cache. This can occur for two reasons:

- The L3 may replace a line due to capacity constraints. If that line has an entry in a WBHT, the L2 cache will not write back the clean line to the L3, but will simply replace it. If the line is accessed again, the request will have to get the line from memory if it cannot be satisfied by another on-chip L2 cache.
- Entries in the WBHT get overwritten due to the small size of the WBHT relative to the total number of cache lines in the system. In this case, clean write backs from the L2 caches will be sent to the L3 even if the line already exists in the L3 cache.

  One additional problem can arise due to the slight lengthening in the time a write back may reside in the write back queue while the WBHT is consulted. This can potentially lead to the queue becoming full. If the write back queue becomes full, misses to the L2 cache will be blocked and will have to wait for an open slot. However, in practice we did not observe this even with a modest write back queue length of eight in the base system.

2.2. Protocol Enhancements

The largest detrimental effect we discovered occurs when memory pressure is low in the system. In this case, writing back all clean lines to the L3 does not hinder performance because there are no contention effects associated with unnecessary clean write backs. Combined with the long access penalty should the WBHT mis-predict whether a line is already in the L3 or not, performance will degrade even with successful WBHT prediction rates of more than 75%. We account for this by using the number of retries on the intrachip bus as an "on/off" switch for the WBHT. We implement a simple timer and maintain a count of retry transactions that occur due to race conditions or resource limitations. When the number of retries in a specified period of time goes below a certain threshold, we do not use the WBHT to make decisions regarding whether or not to write back a clean line, although we do keep the table up-to-date. Instead, all clean write backs are sent to the L3 cache. Surprisingly, a common threshold of two thousand retries every one million processor cycles works well for the applications presented here.

Because of the details of our bus protocol, all L2 caches see the combined snoop response indicating that the L3 cache currently has a valid copy of the line. By having all L2 caches observe this reply, we can place the line's tag in all WBHTs on the chip instead of just the L2 performing the write back. The effects of this optimization are examined in Section 5.

3. Maximizing L2–L2 Transfers

In our assumed CMP architecture, L2 cache–to–cache transfers are considerably faster (by more than a factor of two) than L3 hits. This provides an incentive to maximize L2 misses that are serviced by peer L2 caches. In the base system, L2 write backs (both clean and dirty) are absorbed by the L3 victim cache. In order to reduce latency, we propose allowing other L2 caches to absorb (or *snarf*) write backs when they are able to do so. For this to work, the following issues need to be addressed:

- **Minimize negative interference at the recipient cache:** In order to minimize the chances of an absorbed write back replacing a useful cache line, we need to be selective in choosing candidate lines for replacement in the recipient L2 cache. One obvious choice is invalid lines. Our replacement algorithm first looks for invalid lines. If none are found, we search for lines that are in the "Shared" state. Shared lines are less expensive to replace since, in our coherence protocol, lines will likely only be in that state if another cache has a copy.
- **Ensure that snarfed write backs are reused:** The key to the success of the proposed mechanism is the reuse of write back lines snarfed by other L2 caches. This requires two conditions. First, a means of identifying lines likely to be reused. Second, managing the LRU information at the recipient cache to optimize the chances of such lines staying at the destination until they are reused.
- **Minor modifications to the cache coherence protocol:** The cache coherence protocol needs to be modified to allow multiple L2 caches to signal their ability to absorb a write back. The snoop response generation has to use a fair policy for selecting the cache to receive the line in order to distribute the snarfed write back load to reduce negative interference effects.
- **Minimize cache controller complexity:** Since L2 cache controllers typically do not perform line fills for data that was not requested, the cache controller has been modified to allow this. Instead of resorting to retries on resource conflicts at the potential recipients, we conservatively decline the cache line in that situation.

Since write back line reuse is a significant performance factor for the proposed technique, we ran experiments to understand the frequency of reuse of lines that are written back. Table 2 shows, for each trace used, the percentage of cache lines written back from L2 caches that are later reused both as a percentage of total write backs attempted and write backs accepted by the L3. The table indicates that L2 write back reuse varies across applications, but most applications

Application	% Total	% Accepted
CPW2	27.1	38.4
NotesBench	33.9	53.2
TP	15.5	18.6
Trade2	28.9	58.7

Table 2. Write Back Reuse Statistics

exhibit a significant reuse of L2 write back blocks. This of course makes a good case for having an L3 cache. But in addition, since L2-L2 cache transfers are faster than L3 hits in our system, there is a clear incentive to use any available L2 capacity to keep the lines on-chip in a peer L2 if possible. Analysis of L2 utilization shows that for all applications, there was not enough underutilized space occupied by invalid cache lines that can be used to accept snarfed data. We therefore also victimize lines in peer L2 caches that are in the Shared state because such lines are likely to not be the only cached copy of the line in the system. Conversely, a line in the Exclusive state is guaranteed to be the only valid copy on-chip, and is therefore not a logical choice for replacement. Modified lines are problematic since replacing them would force another write back, resulting in additional traffic.

Another important optimization to improve the effectiveness of the proposed technique is to be selective in choosing the lines that are snarfed. If every write back from an L2 cache were snooped by all L2 caches on the chip, the increase in pressure on the L2 tags will likely offset any performance gains we hope to achieve. To reduce write backs that must access peer L2 cache tags, we added another table (separate from the write back history table) to track cache lines that were written back and later reused by an L2 cache, in effect tracking lines that have been missed on multiple times. As with the WBHT, this table is organized as a cache that maintains the tags of lines that have been replaced, with an additional bit per entry specifying when the line has been missed on either locally or by another L2 cache. The tag for a line is entered into the table when the line is written back by any L2 cache. If the line is later missed on, and the line still has an entry in the table, the "use bit" is set indicating that the line has been previously replaced and subsequently missed on. When such a line is written back again, the lookup table is consulted, and on a hit with the reuse bit set, a special bus transaction bit is set to trigger the snarf algorithm at snooping L2 caches.

The cache controllers and the cache coherence protocol have to be modified slightly to allow an L2 cache to accept a write back by another L2 cache. The cache controllers themselves were modified to include logic similar to that used in the L3 controller to accept write backs. However, unlike the L3 where only a single cache may accept a write back, multiple L2's that snoop a write back might decide to accept it. Their acceptance is signaled using a special snoop reply. Note that if an L2 cache finds that it already has the line in a clean state, the bus response will cause the write back to be squashed in a manner similar to the baseline L3 protocol described in Section 2. In our system, a central entity, referred to as the "Snoop Collector", monitors snoop responses from all bus agents in order to determine the final snoop response. This process occurs in the base coherence protocol implementation. The Snoop Collector was modified to deal with the write back snarf acknowledgments from multiple L2 caches to choose a winner in a fair round-robin fashion from the set of L2 caches that are able to accept the cache line. Since the final snoop response is seen by all bus agents in our system, caches that were willing to accept write backs but did not win are able to deallocate any buffers or other resources that had to be reserved in order to avoid deadlocks.

4. Experimental Methodology

We evaluate the performance impact of the architectural extensions and policies described in Sections 2 and 3 using the Mambo [3] PowerPC full system simulation environment. We simulate the system described in Section 1 using this simulation environment on all four industry-standard commercial workloads.

4.1. Mambo Simulation Environment

For the four commercial workloads examined here, we have L2 cache traffic traces captured on a real SMP machine running the full workloads. We feed the traces into the Mambo cache hierarchy simulator. The cache hierarchy, interconnection network, coherence protocol, and memory subsystem are modeled in detail, including accurate queuing, contention, and timing (Table 3 shows important system parameters and contentionless access latencies). The coherence protocol implemented is an extension of the one used in IBM's POWER 4 systems, which supports both shared and dirty cache-to-cache transfers [13]. One parameter we vary is the maximum number of outstanding read and write misses per thread that can be simultaneously present in the system at a time. This parameter would be determined in real systems by either the number of entries in the load/store queue in a superscalar processor, the number of MSHRs supported by the cache hierarchy, the number of concurrent threads supported by each core, and/or the application(s) being executed. Changing this simulation parameter serves to increase or decrease the memory pressure on the system. Finally, all traces contain both application and OS references, and therefore give a more realistic picture of the characteristics of the applications executed.

Processors	8, 2-way SMT
Frequency	6 GHz
L2 Size	4 slices, 512 KB each
Number of L2 Caches	4
L2 Associativity	8-way
L2 Latency	20 cycles
L2-to-L2 Transfer Latency	77 cycles
L3 Size	4 slices, 4 MB each
L3 Associativity	16-way
L3 Latency	167 cycles
Bi-directional Ring Bus	1:2 core speed, 32B-wide
Memory Controller	1:2 core speed
Memory Latency	431 cycles (from core)

Table 3. System Parameters

4.2. Applications

We used traces from four commercial, industry-standard workloads as described below.

Transaction Processing (TP): This workload models an online transaction processing system that involves a similar mix of transactions and database properties as in the TPC-C benchmark from the Transaction Processing Council [14]. TPC-C benchmark runs require strict audited adherence to a set of rules which is not true of our TP workload. The TP workload was tuned to yield over 92% CPU utilization.

CPW2: The Commercial Processing Workload (CPW) application simulates the database server of an online transaction processing (OLTP) environment and tests a range of database applications, including simple and medium complexity updates, simple and medium complexity inquiries, realistic user interfaces, and a combination of interactive and batch activities. In particular, while the transaction processing (TP) application maintains an extremely high load on the CPU, CPW2 is designed to be measured at approximately 70% CPU utilization in order to avoid misrepresenting the capacity of larger systems. CPW2 also uses database and system values that better represent the way the system is shipped to IBM customers. More information on this workload is available [8].

NotesBench: NotesBench [11] is a workload for evaluating email server performance. NotesBench simulates an average group of users performing everyday mail tasks while connected to a Domino Lotus Notes mail server. The workloads cover a variety of protocols such as IMAP, NRPC (Notes Remote Procedure call, which is Domino/Notes native mail), and HTTP.

Trade2: Trade2 is an end-to-end Web application modeled after an online brokerage. Trade2 leverages J2EE components such as servlets, JSPs, EJBs, and JDBC to provide a set of user services through the HTTP protocol. The specific services modeled in Trade2 enable users to register for account and profile establishment, log into an account with session creation after validation, retrieve the current account balance and market conditions, modify the user profile, obtain security quotes and purchase stock establishing a new portfolio holding, viewing the portfolio, selling portfolio holdings, and logging off. More information on Trade2, including details on the EJB implementation and the database schema used is available [7]. Sample code for the Trade2 application is available for download [6].

5. Performance Evaluation

In this section we describe the performance improvements of utilizing the write back history table to reduce the number of unnecessary write backs and allowing write backs from L2 caches to be absorbed by peer on-chip L2 caches. All results are presented relative to the baseline system described in Section 1 where all clean and dirty lines replaced in the L2 cache are sent to the L3 cache. This baseline configuration does filter lines written back from the L2 if the line appears in the L3 cache by having the L3 cache squash the initial write back request after it is snooped.

5.1. Results for Write Back History Table

Figure 2 shows the percentage performance improvement in execution time relative to the base configuration of the four commercial applications when a 32K-entry WBHT is used. We present results for varying numbers of allowed outstanding load miss requests per thread (four threads feed each L2 cache in the system). Note that the actual performance improvement is a factor of both the number of write back notifications the WBHT can eliminate as well as the current level of memory pressure. As the maximum number of outstanding loads is increased, more pressure is exerted on the memory hierarchy. Table 4 gives relevant statistics comparing the base protocol with the system enhanced with the WBHT for six outstanding loads per thread. As depicted in Figure 2, because the WBHT reduces intranode bus utilization by eliminating unnecessary write backs, it has a greater impact on performance with increasing memory pressure.

Table 4 also gives the percentage of times the WBHT made “the right decision” on whether to write back a clean line or not. We measure this by peeking into the L3 cache in the simulator when the WBHT makes its decision and counting whether the decision was correct or not. In general, the WBHT correctly predicts whether a line is resident in the L3 cache between 60% and 75% of the time. For situations where the L3 hit rate is critical to performance (e.g., low memory subsystem contention and high data reuse, as in the TP application), these levels of misprediction may hinder performance, leading to our use of

	CPW2		NotesBench		TP		Trade2	
	Base	WBHT	Base	WBHT	Base	WBHT	Base	WBHT
WBHT Correct	N/A	63.1%	N/A	67.3%	N/A	75.3%	N/A	60.4%
L3 Load Hit Rate	50.5%	37.3%	70.5%	70.4%	32.4%	25.4%	79.0%	67.8%
L2 Write Back Requests	73M	50M	31M	30M	88M	70M	133M	64M
L3-issued Retries	3.0M	2.6M	.24M	.24M	66M	63M	2.0M	1.5M

Table 4. Effects of Write Back History Table (6 Loads per Thread Maximum)

the retry counter as a switch for the WBHT (see Section 2). Therefore, with low memory pressure (one or two outstanding loads per thread), the small retry rate ensures that the WBHT is not invoked, and most applications show no performance improvement or degradation, with the exception of TP. At two outstanding loads per thread, the retry rate in TP is barely high enough to exceed our threshold, triggering the use of the WBHT. However, Table 4 indicates that TP has a a very low L3 hit rate, which the WBHT further reduces from 32.4% to 25.4%, causing performance to dip as shown in Figure 2.

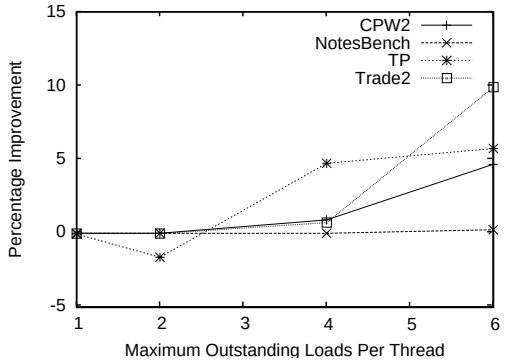

Figure 2. Runtime Improvement Over Baseline of Write Back History Table

At six outstanding loads per thread, all applications with the exception of NotesBench show marked improvement over the baseline protocol. Of course, the real performance of each of these applications likely lies somewhere on the lines plotted in Figure 2 depending on system details. In Trade2, despite the L3 hit rate dropping from 79% to 68%, the large reduction in L2 write back requests and subsequent decrease in the L3 retry rate allows Trade2 to perform well in our system. In contrast, the WBHT is almost never invoked for the NotesBench application due to the very low demands placed on the memory subsystem by this application. CPW2 shows modest improvement with increasing memory pressure.

Figure 3 shows the performance increase over the baseline system when we allow all L2 caches to allocate entries in their WBHT simultaneously, in contrast to Figure 2 in which only the L2 cache attempting the write back would do so. The trends are basically the same for each application. In this experiment, all L2 caches snoop the combined bus response indicating that the L3 has a clean line already. Thus, an entry is allocated in all WBHT's instead of just the one attempting the write back. This global distribution provides a small increase for all applications when memory contention is high, with Trade2 benefiting the most (a 2% increase over the previous results).

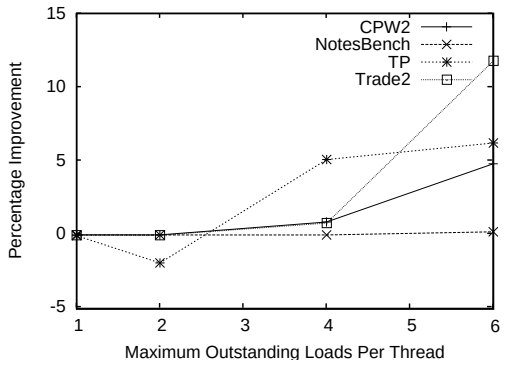

Figure 3. Runtime Improvement of Updating All WBHTs Using L3 Snoop Response

Figure 4 plots the improvement in performance, normalized to the performance with a 512-entry WBHT, for our workloads as the size of the WBHT is varied from 1K entries to 64K entries. We can see that all applications generally perform better as the history table increases in size, although the performance of NotesBench, CPW2, and TP grows much slower than that of Trade2. If lines that are repeatedly written back from the L2 and then subsequently missed on can be kept in the WBHT, its effect on performance will be much more pronounced. As the table grows,

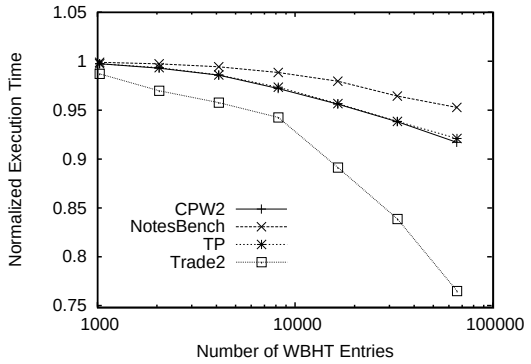

Figure 4. Normalized Runtime of Varying L2 WBHT Sizes Normalized to 512-Entry WBHT System

more of these lines are kept longer as the conflict rate in the table decreases. Eventually, tags for lines remain in the WBHT so long that the history table gets out of sync with the actual contents of the L3, leading to a leveling off in performance with very large table sizes (larger than those shown in Figure 4). Statistics show that many lines in Trade2 are written back and then re-referenced more than 300 times, whereas few lines in CPW2 show this behavior more than 20 times, leading to the discrepancy in the performance gain between the two applications.

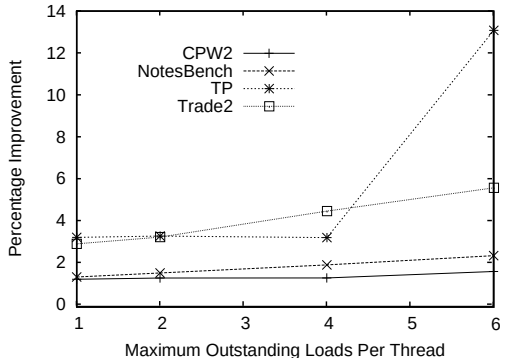

Figure 5. Runtime Improvement Over Baseline of Allowing L2 Snarfing

5.2. Results for L2-to-L2 Write Backs

Figure 5 shows the performance improvement of the four commercial applications relative to our baseline configuration when using an L2-to-L2 history table containing 32K entries. In the L2-to-L2 write back case, L2 caches snoop write backs from peer L2 caches and absorb them if possible. Additionally, if a peer L2 cache snoops a write back request, and the line is already valid in the peer L2, the actual write back operation is squashed through a special snoop reply.

Recall that L2-to-L2 write backs replace invalid or potentially useful shared cache lines in the recipient L2 in an attempt to convert L3 or memory accesses into L2-to-L2 transfers. Our concern that using space in peer L2 caches for write backs would lead to worse hit rates for those L2 caches absorbing the write backs proved unfounded. Table 5 shows that for all applications, the use of L2-to-L2 write backs was not detrimental to local (i.e., snarf recipient) L2 cache hit rates. In fact, all four applications showed improved L2 hit rates, although the improvement was slight for the majority of the applications. This improvement in hit rate can be attributed to snarfed lines being used at the recipient cache, effectively performing a useful prefetch.

As seen in Figure 5, the performance improvement for CPW2 and NotesBench remains relatively flat regardless of changes in the memory pressure. Table 5 indicates that for NotesBench, the 2.4% performance improvement over the base case arises from a large reduction in the number of L3 retries (94%), and a slight reduction in the number of off-chip accesses (1.1%). The reduction in L3 retries for all applications occurs because lines being written back are frequently found in peer L2 caches. The squashing of these write backs reduces the occupancy of requests sitting in the L3 queues, thereby reducing the retries the L3 must issue. For NotesBench, 6% of snarfed lines were used to satisfy requests from a thread associated with the snarfing L2, while 13% of snarfed lines were used to satisfy data requests by other L2 caches on the chip. CPW2 shows similar trends to NotesBench for L2-to-L2 transfers.

Trade2 shows a 5.9% performance improvement with high numbers of outstanding loads. Trade2 has the highest reduction in off-chip accesses of the four workloads used here (Table 5), as well as the highest increase in local L2 hit rates.

The sharp rise in performance with L2 snarfs for the TP workload results from a dramatic decrease in retry responses generated by the L3 cache. Over 99% of all L3 retries, of which there are a relatively large number (see Table 4), are eliminated. Additionally, 16% of the snarfed lines are used by the snarfing L2 to satisfy cache accesses, a higher rate than any other application studied, even though the overall reduction in off-chip accesses is the lowest of

	CPW2	NotesBench	TP	Trade2
Performance Improvement	1.7%	2.4%	13.1%	5.6%
Reduction in Off-Chip Accesses	1.2%	1.1%	.8%	5.2%
Write Backs Snarfed	3.7%	2.5%	2.8%	7.0%
Snarfed Lines Used Locally	10%	6%	16%	4%
Snarfed Lines Provided for Interventions	16%	13%	14%	10%
Increase in Local L2 Hit Rate	.4%	1.2%	.3%	3.7%
L3-Issued Retry Rate Reduction	96%	94%	99%	93%

Table 5. Effects of L2-to-L2 Write Backs (6 Loads Per Thread Maximum)

any application. Given that the TP workload typically exhibits high CPU utilization in practice, this level of memory pressure may never be experienced in the actual execution of the application. Discovering the correct level of memory pressure for each of these applications for use in future system design is the subject of ongoing work.

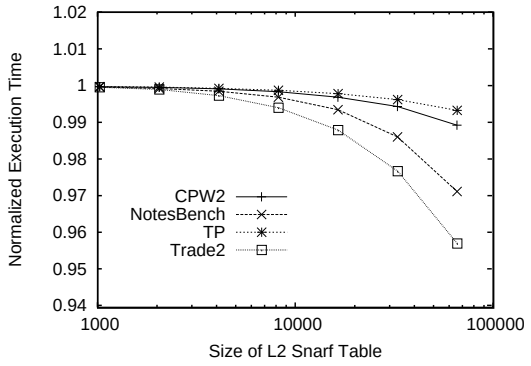

Figure 6. Runtime of Varying L2 Snarf Table Sizes Normalized to 512-Entry Snarf Table System

Figure 6 shows the performance improvement trend as the number of entries maintained in the L2-to-L2 transfer table is varied when six outstanding load misses per thread are allowed. The results show that the interaction between the size of this table and performance improvements varies for the applications studied. When the table is too small, the L2 controllers cannot maintain enough history, so some opportunities for improvement are missed. On the other hand, as the L2 snarf table size increases, the possibility of stale data affecting the correctness of write back decisions is increased. Overall, the size of the L2 snarf table has little impact on the effectiveness of the mechanism beyond a certain point, with Trade2 again showing the most sensitivity to the size of the table. However, the performance of this application improves by only 4.5% over the minimal table size, even with a very large table size of 64K entries.

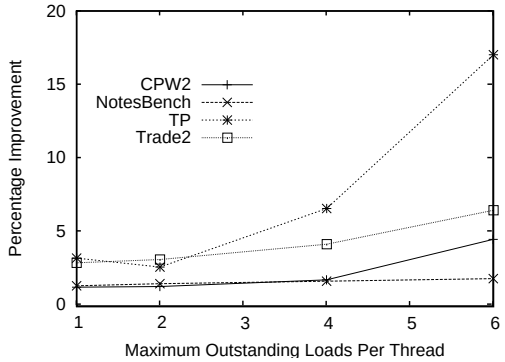

Figure 7. Runtime Improvement Over Baseline of Combined Tables

5.3. Interaction

Finally, we examined the effect on performance of combining the write back history table and the L2 snarf table. In the combined case, we use two tables, but to preserve the overall space requirements, each table is only 16K entries as opposed to 32K entries as used in Sections 5.1 and 5.2. Figure 7 shows that the performance benefits of the individual optimizations are not additive. Considering Trade2, which showed the most promising results for L2-to-L2 write backs, the combined version improved by less than the selective write back alone when the memory pressure is high. When the pressure is low, the performance afforded by snarfing L2 write backs allows the runtime to improve over that of the WBHT-only system. Keep in mind that while most applications showed little sensitivity to the L2 snarf table, all applications benefit from a larger write back history table. The benefits of combining the two schemes allow

TP to perform better than either the WBHT or L2 snarf table alone, despite the fact that the tables are half the size here as in Sections 5.1 and 5.2.

6. Related Work

Norman Jouppi [10] first proposed to use a fully-associative victim cache to improve the performance of direct-mapped caches. Since then, victim caching has been the subject of many research projects, including designs of victim caches for vector registers [4] and the use of victim caches to reduce power [12]. The majority of these projects focus on improving the performance of L1 caches. All of the projects that we are aware of use victim caches that are closely positioned to the main caches. This study focuses on the use of a large, off-chip L3 victim cache for clean and dirty lines in a CMP system.

Bhandarkar and Ding [2] show that L2 miss penalty has the strongest negative effect in CPI (Cycles Per Instructions). To speed up the handling of L2 misses, many architectures are incorporating another level of caches (L3 caches) between L2 and memory. For instance, Intel Itanium2 [9] is equipped with an on-die L3 cache up to 6MB. IBM POWER4 [13] allows each dual-processor chip to share a 32MB L3 cache, whose directory is on the chip but whose data is off the chip.

The Piranha system [1] attempts to improve cache utilization by not supporting inclusion between the L1 and L2 caches and allowing forwarding of requests from L2 to L1 caches.

Even though L3 caches are becoming more common, only a limited number of studies have been done on how to best exploit them to reduce memory latency. Ghai *et al*. [5] demonstrate that an 8-way 4MB L3 shared by four processors decreases the average memory access time by 6.5% and reduces the traffic by 95% from a system without an L3. In their experiments, the L3 cache maintains inclusion with the caches above it. While the L3 cache may filter out some snoop traffic to the L2 cache in a multi-CMP system, it increases the snooping delay for cache-to-cache transfers. To maintain inclusion, whenever the L3 replaces a line it must broadcast an invalidation request to all L2 caches. In addition, supporting inclusion requires replicated copies to be stored in the L2 and L3 caches simultaneously, causing inefficient use of L2 and L3 cache space. For these reasons we did not choose to support inclusion for our study.

7. Conclusions and Future Work

In this paper, we have examined both architectural and policy decisions regarding the use of a large, shared L3 cache in the context of a chip multiprocessor. We believe that the present-day trend of placing more cores with support for more independent hardware threads on a single CMP will lead to increased pressure on the cache hierarchy. In such a situation, managing all aspects of cache interactions is important. Here we have shown that simple, adaptive mechanisms to more intelligently manage write back traffic can have a positive impact on performance.

We have proposed the use of a small hardware table to provide hints to L2 caches as to which lines are present in a lower-level L3. This write back history table serves to filter the writing back of clean lines from the L2 cache when there is a good chance that these lines are already present in the L3 cache. Our experiments with four commercial workloads show that a small history table, on the order of 10% or less than the size of the L2 cache, can remove more than 50% of such “unnecessary” clean write backs. Depending on the memory pressure, this can lead to an up-to 13% performance improvement.

We have also evaluated allowing L2 write backs of lines believed to be candidates for reference in the near future to be placed in peer on-chip L2 caches. The results of this optimization varied across the applications studied with most applications showing some improvement. We saw reductions in off-chip accesses and L3 retry rates for all applications, and the lines “snarfed” by peer L2 caches were used to both satisfy local requests and interventions.

Currently, we are investigating alternate L3 organizations and policies, including having separate buses for chip-private L3 caches and memory, similar to the POWER 5 architecture from IBM. One idea we are investigating for reducing the size of the WBHT presented here is to allow each entry in the table to serve multiple cache lines, reducing the size of each entry and providing greater coverage at the risk of increased prediction errors. Finally, we are developing new replacement algorithms that take into account information contained in the history tables presented here to better utilize all available cache space.

Acknowledgments

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract No. NBCH30390004. Details presented in this paper may be covered by existing patents or pending patent applications.

References

[1] L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 282–293, June 2000.

[2] D. Bhandarkar and J. Ding. Performance Characterization of the Pentium Pro Processor. *Proceedings of the 3rd IEEE Symposium on High Performance Computer Architecture*, pages 288–297, February 1997.

[3] P. Bohrer, M. Elnozahy, A. Gheith, C. Lefurgy, T. Nakra, J. Peterson, R. Rajamony, R. Rockhold, H. Shafi, R. Simpson, E. Speight, K. Sudeep, E. V. Hensbergen, , and L. Zhang. Mambo - A Full System Simulator for the PowerPC Architecture. *ACM SIGMETRICS Performance Evaluation Review*, 31(4), March 2004.

[4] R. Espasa and M. Valero. A Victim Cache for Vector Registers. *International Conference on Supercomputing*, pages 293–300, July 1997.

[5] S. Ghai, J. Joyner, and L. John. Investigating the Effectiveness of a Third Level Cache. *Technical Report TR-980501-01, Laboratory for Computer Architecture, The University of Texas at Austin*, May 1998.

[6] IBM. Websphere Performance Benchmark Sample. *http://www.ibm.com/software/webservers/appserv/wpbs_download.html*.

[7] IBM. Application Development Using the Versata Logic Suite for Websphere. *Redbook SG24-6510-00, Available from http://www.redbooks.ibm.com*, December 2002.

[8] IBM. iSeries Performance Capabilities Reference V5R2. *Available from http://publib.boulder.ibm.com/pubs/html/as400/online/chgfrm.htm*, 2003.

[9] Intel Corporation. Intel Itanium-2 Processor Specification Update. *Document Order Number 249634*, July 2004.

[10] N. P. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully Associative Cache and Prefetch Buffers. *Proceedings of the 17th International Symposium on Computer Architecture*, pages 364–375, June 1990.

[11] Lotus NotesBench Consortium. NotesBench Description. *Available from http://www.notesbench.org*.

[12] G. Memik, G. Reinman, and W. Mangione-Smith. Reducing Energy and Delay Using Efficient Victim Caches. *International Symposium on Low Power Electronics and Design*, pages 262–265, August 2003.

[13] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy. POWER4 System Microarchitecture. *IBM Journal of Research and Development*, 46(1):5–26, January 2002.

[14] Transaction Processing Performance Council. TPC Benchmark C Standard Specification. *Available from http://www.tpc.org/tpcc/spec/tpcc_current.pdf*.