Quantifying and Reducing the Effects of Wrong-Path Memory References in Cache-Coherent Multiprocessor Systems

Resit Sendag[1], Ayse Yilmazer[1], Joshua J. Yi[2], and Augustus K. Uht[1]

[1] - Department of Electrical and Computer Engineering
University of Rhode Island
Kingston, RI
{sendag,yilmazer,uht}@ele.uri.edu

[2] - Networking and Computing Systems Group
Freescale Semiconductor, Inc.
Austin, TX
joshua.yi@freescale.com

Abstract

High-performance multiprocessor systems built around out-of-order processors with aggressive branch predictors execute many memory references that turn out to be on a mispredicted branch path. Previous work that focused on uniprocessors showed that these wrong-path memory references may pollute the caches by bringing in data that are not needed on the correct execution path and by evicting useful data or instructions. Additionally, they may also increase the amount of cache and memory traffic. On the positive side, however, they may have a prefetching effect for memory references on the correct path. While computer architects have thoroughly studied the impact of wrong-path effects in uniprocessor systems, there is no previous work on its effects in multiprocessor systems. In this paper, we explore the effects of wrong-path memory references on the memory system behavior of shared-memory multiprocessor (SMP) systems for both broadcast and directory-based cache coherence. Our results show that these wrong-path memory references can increase the amount of cache-to-cache transfers by 32%, invalidations by 8% and 20% for broadcast and directory-based SMPs, respectively, and the number of writebacks by up to 67% for both systems. In addition to the extra coherence traffic, wrong-path memory references also increase the number of cache line state transitions by 21% and 32% for broadcast and directory-based SMPs, respectively. In order to reduce the performance impact of these wrong-path memory references, we introduce two simple mechanisms – filtering wrong-path blocks that are not likely-to-be-used and wrong-path aware cache replacement – that yield speedups of up to 37%.

1 Introduction

Shared-memory multiprocessor (SMP) systems are typically built around a number of high-performance out-of-order superscalar processors, each of which employs aggressive branch prediction techniques in order to achieve high issue rate. During program execution, these processors speculatively execute the instructions following the target of a predicted branch instruction. When a branch is mispredicted, the processor must restore its state to the state that existed prior to the mispredicted branch before the processor can start executing instructions down the correct path. However, during speculative execution, *i.e.*, before the branch outcome is known, the processor speculatively issues and executes many memory references down the wrong-path. Although these wrong-path memory references are not allowed to change the processor's architectural state, they do change the data and instructions that are in the memory system, which can affect the processor's performance.

Previous work [1, 5, 8, 13-17, 19-22] studied the effects that speculatively executed memory references have on the performance of out-of-order superscalar processors. These papers yield several conclusions. First, wrong-path memory references may function as indirect prefetches by bringing data into the cache that are needed later by instructions on the correct execution path [14, 19, 20, 21]. Unfortunately, these wrong-path memory references also increase the amount of memory traffic (*i.e.*, increased bandwidth consumption) and can pollute the cache with cache blocks that are not referenced by instructions on the correct path [13, 14, 19, 22]. Of these two effects, cache pollution – particularly in the L2 cache – is the dominant negative effect [13, 14]. The results in [14] also show that it is extremely important to model wrong-path memory references, since they have a significant impact on the estimated performance.

In this paper, we focus on the effect that wrong-path memory references have on the memory system behavior of SMP systems, in particular, for both broadcast-based and directory-based cache coherence. For these systems, not only do the wrong-path memory references affect the

performance of the individual processors, they also affect the performance of the entire system by increasing the number of cache coherence transactions, the number of cache line state transitions, the number of writebacks and invalidations due to wrong-path coherence transactions, and the amount of resource contention (buffer usage, bandwidth, *etc.*).

In this paper, to minimize the effect that wrong-path memory references have on the performance of a SMP system, we propose and evaluate a simple mechanism to filter out the wrong-path cache blocks that are unlikely to be used on the correct-path. Our filtering mechanism uses temporal locality and L1 data cache evictions to determine if the corresponding cache block should be evicted from the L2 cache. In addition to this filtering mechanism, we also propose a cache replacement policy that is wrong-path aware. More specifically, we add a field to each cache line to indicate whether or not that cache line was due to an instruction on the correct-path or the wrong-path. When evicting a cache block from a set, evict the oldest wrong-path cache block. Our results show that both of these simple mechanisms can significantly reduce the negative impact that wrong-path memory accesses have on the performance of SMP systems.

This paper makes the following contributions:

1. It analyzes and quantifies the effect that wrong-path memory accesses have on the performance of SMP systems, in particular, how wrong-path memory accesses affect the cache coherence traffic and state transitions, and the resource utilization.
2. It proposes a filtering mechanism and a replacement policy that evicts the oldest wrong-path cache blocks first to minimize the impact that wrong-path memory references have on the performance of SMP systems.

The remainder of the paper is organized as follows – Section 2 describes the effects that wrong-path memory references can have on the memory system behavior of SMP systems. Sections 3 and 4 present the details of the simulation environment and the simulation results, respectively. Section 5 describes our filtering mechanism and the wrong-path aware replacement policy, and how they reduce negative effects of wrong-path memory references. Section 6 describes some related work, while Section 7 concludes and suggests some future work.

2 Wrong-Path Effects

When designing a coherent shared-memory interconnect, the most important design decision is the choice of the cache coherence protocol. Popular protocols include: MSI (Modified, Shared, Invalid), MESI (Modified, Exclusive, Shared, Invalid), MOSI (Modified, Owned, Shared, Invalid), and MOESI (Modified, Owned, Exclusive, Shared, Invalid) [9]. Since the cache coherence protocol maintains the illusion of sequential consistency between processors, when a processor accesses memory, the coherence state (*i.e.*, M, O, E, S, or I) of the cache lines in the processors' data caches may change. However, although the branch prediction accuracy of modern high-performance processors is high, when a branch misprediction does occur, loads on the mispredicted path access the memory subsystem, which generates additional coherence traffic (additional communication and state transitions). While these extra state transitions do not violate the illusion of sequential consistency, they may degrade the performance of the cache coherence protocol and, subsequently, the performance of the memory subsystem, and, finally, the performance of the SMP. In the remainder of this section, we discuss the potential effects that wrong-path memory references can have for each of the aforementioned four cache coherence protocols (MSI, MESI, MOSI, and MOESI).

2.1 Replacements

A speculatively-executed load instruction that is later determined to be on a mispredicted path may bring a cache block into data cache that replaces another block that may be needed later by a load on the correct-path. As a result of these replacements, wrong-path loads pollute the data cache [13, 19], which may cause additional cache misses. Figure 1, Step 2 shows an example of this situation. In this example, Processor 0 speculatively requests Block A, which causes the replacement.

On the other hand, these speculatively accessed memory references can potentially hide the memory latency for later correct path misses, *i.e.* prefetching [14, 19-21], which can improve the processor's performance.

2.2 Writebacks

In contrast to the writebacks caused by the correct-path replacements, in a SMP system, the coherence actions caused by wrong-path memory references can also cause writebacks. For example, if the requested wrong-path block has been modified by another processor, *i.e.*, its cache coherence state is M, a shared copy of that block is sent to the requesting processor's cache, which subsequently may cause a replacement. When the evicted block has a cache coherence state of M (exclusive, dirty) or O (shared, dirty) state, this causes an additional writeback, which would not have occurred if the wrong-path load had not accessed memory in the first place. Step 2 in Figure 1illustrates this example. Extra writebacks, in addition to what is discussed above, may occur in MSI or MESI coherence SMPs. For these two protocols, if the requested wrong-path block is in the M state in

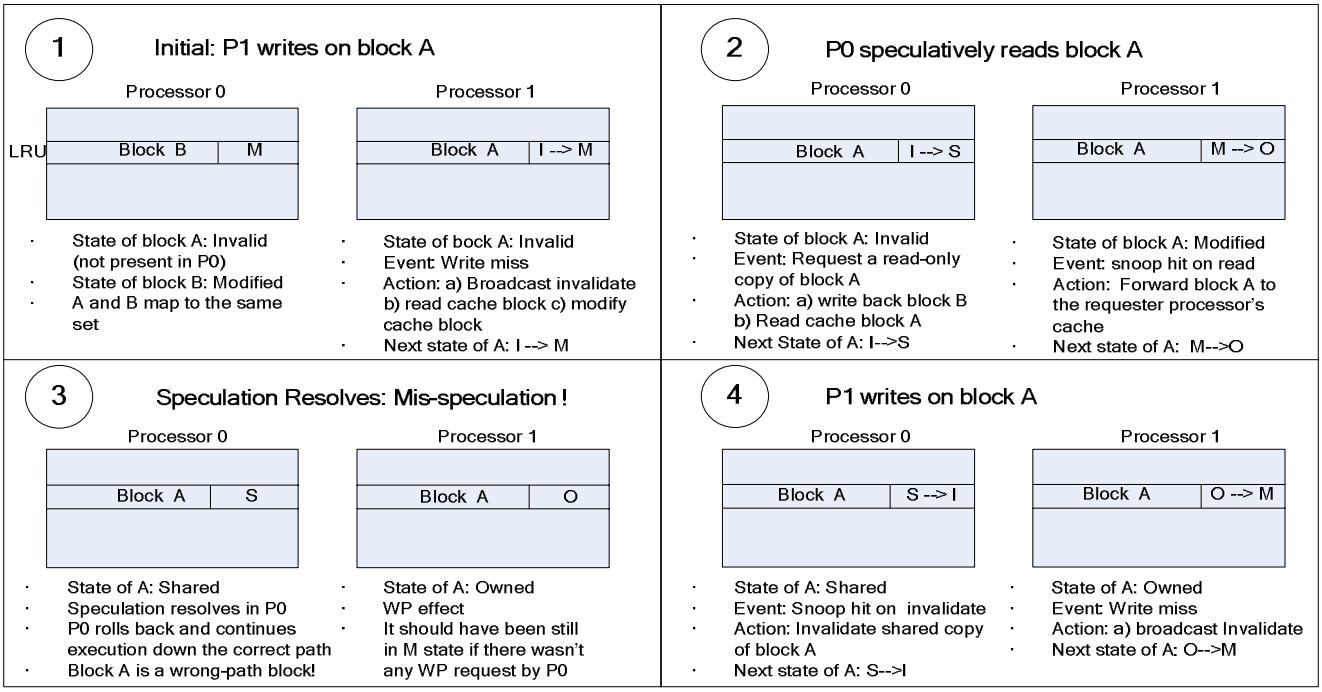

Figure 1. Summary of the wrong-path effects on a SMP system for MOSI (Modified, Owned, Shared, Invalid) or MOESI (Modified, Owned, Exclusive, Shared, Invalid) coherence protocols. Blocks A and B map to the same cache. (1) Initially, block B is in the Modified (M) state in P0's cache and it is the LRU (Least Recently Used) block in the set, while block A is in P1's cache in the M state. (2) P0 speculatively reads block A. A Shared (S) copy of the block replaces block B and causes a writeback. The copy in P1's cache changes its state to O. (3) Speculation turns out to be incorrect. Note the extra cache transactions and state transitions. (4) P1 writes on block A and gets the exclusive ownership (state of block A is M now). This causes invalidation to be sent to the caches sharing block A.

another processor's cache, a shared copy of that block is sent to the requesting processor's cache and also it is written back to the memory. Then the cache coherence state of that cache block is demoted from M to S in the original owner's cache. This additional writeback may not occur without the wrong-path load.

2.3 Invalidations

To maintain the illusion of sequential consistency, a store instruction cannot write its value to memory until it becomes the oldest (*i.e.*, non-speculative) instruction in the processor. As a result, store instructions can never cause any extra invalidations.[1] However, wrong-path loads may cause additional invalidations.

For example, assuming a MOESI protocol, when a wrong-path load instruction accesses a cache block that another processor has modified, the state of that cache block changes from M to O in the owner's cache and will have a cache coherence state of shared, S, in the requester's cache. If the owner of that cache block needs to write to it, the owner changes the state of that block from O to M and invalidates all other copies of that cache block. Therefore, as this example shows, changes in the cache coherence state of a cache block due to a wrong-path load can cause additional invalidations. Figure 1, Step 4 illustrates this example.

2.4 Cache Block State Transitions

In addition to causing additional replacements, writebacks, and invalidations, wrong-path memory references can also cause transitions in the cache coherence state of a cache block. For example, when a wrong-path memory reference accesses a modified cache block in another processor's cache, under the MOESI protocol, the cache coherence state of that block changes from M to O in the owner's cache. The state of that cache block changes back to M when the owner writes to that block. These changes in the cache coherence is due solely to the wrong-path access. Therefore, in this case, a wrong-path memory access in another processor results in two extra cache state transitions in the owner's cache (see Steps 2 and 4 in Figure 4).

[1] In this study, we did not consider the multiprocessors based on uniprocessors that may speculatively execute store instructions, such as the speculative versioning cache of Multiscalar [32], or Levo [34].

Table 1. Benchmarks and input data sets

Benchmark	Description	Input Data Set
fft	Complex 1-D FFT	64K points
radix	Integer radix sort	2M integers, radix 1024
ocean	Simulates large-scale ocean movements	128x128 ocean
water-spatial	Simulation of water molecules	512 molecules
em3d	Electromagnetic force simulation	400K nodes, degree 2, span 5, 15% remote

Table 2. Broadcast (snoop)-based and directory-based SMP system parameters

Parameter	Value
Processors	16 UltraSPARC III processors
Processor Parameters	2 GHz 15-stage pipeline, out-of-order execution
	8-wide dispatch/retirement
	256/128-entry ROB/scheduler
	10 cycle branch misprediction penalty
	GSHARE branch predictor with 4K PHT
	64-entry return address stack
	32 Entry CAS and CAS exception table
L1 Caches	Split I/D, 32KB 2-way, 128 Byte Blocks, with 2ns access latency
	32 Entry MSHRs
L2 Caches	Unified, 2 MB 2-way, 20ns hit latency
	Exclusive L1 and L2s
Main Memory	4 GByte per bank, 240ns DRAM latency
Interconnect	Hierarchical Switch

2.5 Data/Bus Traffic and Coherence Transactions

Due to these extra replacements, writebacks, invalidations, and changes in the cache coherence state, wrong-path memory accesses increase the amount of traffic due to L1 and L2 cache accesses, as well as increasing the number of snoop and directory requests.

2.6 Power Consumption

In the best case, even if wrong-path memory references do not affect the performance of the SMP system, they still may increase system's overall power consumption [23].

Several previous studies proposed methods to reduce the power in snoop-based systems [23-27] by filtering unnecessary snoops. In particular, Moshovos *et al*. [23] showed that filtering unnecessary snoops can reduce the total L2 cache power by 30%. Accordingly, reducing the cache line transitions and cache coherence traffic due to wrong-path memory accesses should also reduce the power consumption. However, in this paper, we defer a detailed examination of the attendant power consumption implications to future work.

2.7 Resource Contention

Finally, in addition to the aforementioned effects, wrong-path memory accesses can also increase the amount of resource contention. More specifically, wrong-path memory accesses compete with correct-path memory accesses for the multiprocessor's resources, such as request and response queues at the communication interconnect, and interprocessor bandwidth. The additional cache coherence transactions may increase the frequency of full service buffers, which increases the chance for deadlock. In this paper, however, we assume a sufficient network bandwidth to keep the network contention low. With the possible exceptions of *fft,* which uses all-to-all communication, and *em3d*, network contention was not a problem for the benchmarks that we studied in this paper. However, for other workloads, network contention could have a serious performance impact.

3 Experimental Methodology

3.1 Benchmarks

Table 1 lists the five benchmarks that we used in this paper. The first four benchmarks are benchmarks from the SPLASH-2 benchmark suite [31], while *em3d* [33] is an electromagnetic force simulation benchmark.

3.2 Simulated System Configurations

In this paper, we evaluate a 16-processor SPARC v9 system running an unmodified copy of Solaris 9. We simulate both snoop-based and directory-based SMP systems with an invalidation-based cache coherence. We use the MOSI and MOESI cache coherence protocols, respectively, for the snooping-based and directory-based SMP systems. Each node includes an aggressive, dynamically-scheduled, out-of-order processor core [10], two levels of cache, coherence protocol controllers, and a memory controller [11]. Table 2 lists the relevant parameters of simulated SMP systems.

3.3 Simulation Methodology

We collect our simulation results using the GEMS [10] extension to Virtutech's Simics [35], which is a full system simulator. GEMS adds cycle-accurate models of an out-of-order processor core, cache hierarchy, various cache coherence protocols, multibanked memory (unified or distributed), and various interconnection networks to the base-version of Simics.

To avoid measuring the time needed for thread-forking, we begin our measurements at the start of the parallel phase by using Simics' functional simulation to execute the benchmarks until the start of the parallel phase. Then, we use first iteration of the loop to warm-up the caches and branch predictors. After the first iteration, we simulate the benchmark for one iteration to gather our simulation results.

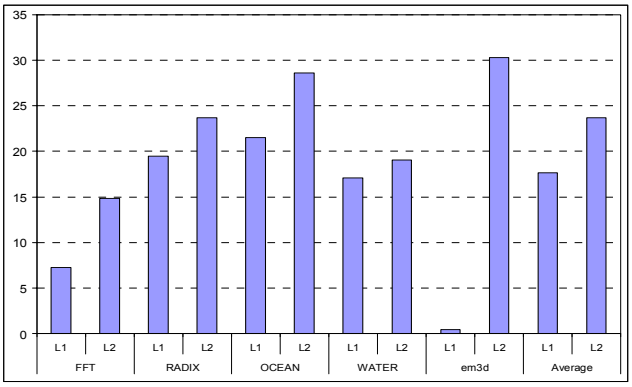

Figure 2. Percentage of increase in L1 and L2 cache traffic for broadcast-based SMPs

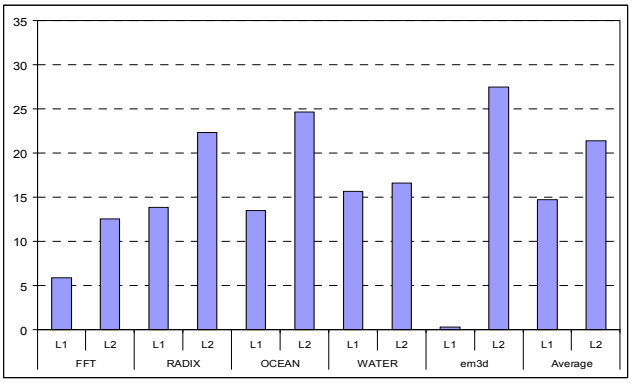

Figure 3. Percentage increase in L1 and L2 cache traffic for directory SMPs

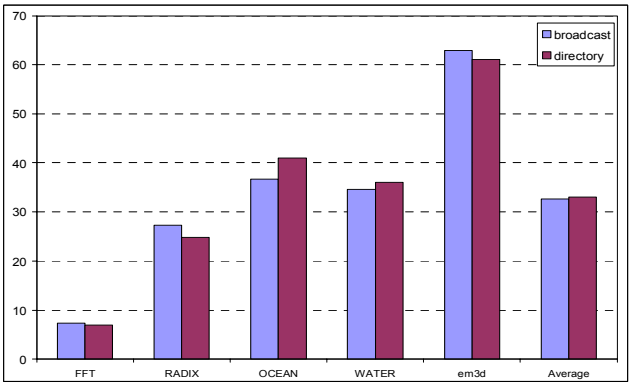

Figure 4. Percentage of increase in coherence traffic for SMPs

4 Evaluating the Wrong-Path Effects

In this section, we evaluate the impact that executing wrong-path memory references have on the caches of the processors in the SMP, the communication between processors due to coherence transactions, and the overall performance of SMP. To measure the various wrong-path effects, we track the speculatively generated memory references and mark them as being on the wrong-path when the branch misprediction is known.

4.1 L1, L2, and Coherence Traffic

Figure 2 shows the increase in the traffic between processor and its L1 data cache due to wrong-path memory references, as a percentage of the total number of memory references, for both broadcast and directory-based SMPs. Figure 3 does the same for the L1 data cache and L2 cache traffic.

Figure 2 shows that wrong-path loads increase the total number of memory accesses by an average of 17% and 14%, for broadcast and directory-based SMPs, respectively, while Figure 3 shows that these loads increase the percentage of L2 cache accesses by 23% and 21% for broadcast and directory-based SMP systems, respectively. For all benchmarks and for both SMP systems, the percentage increase in the number of L2 references is larger than the percentage increase in the number of L1 cache references. For *em3d*, while this increase is negligible for both systems, the number of L1 misses increases by as much as 30%.

Figure 4 shows that wrong-path memory accesses increase the number of coherence transactions by an average of 32% and 33% for broadcast and directory-based SMPs, respectively, while for *em3d*, the coherence traffic increases by over 60%.

4.2 Cache Line Replacements

Wrong-path memory references can have both a positive and negative effect on the processor's performance by either prefetching data into the caches or by polluting them [13, 14, 19, 20], respectively. To determine their performance impact, we categorize the misses caused by wrong-path loads into four groups: *unused*, *used*, *direct miss*, and *indirect miss*. We classify a wrong-path cache block as *unused* when it is evicted before being used or is never used by a load on the correct-path. Conversely, we classify a wrong-path cache block as *used* when a correct-path memory access references it. A *direct miss* cache block replaces a cache block that will be referenced later by a correct-path load, which has a very large performance impact, but is itself evicted before being used.

Finally, one side-effect that *unused* wrong-path cache blocks have is that they change the LRU state of the other cache blocks in that set, which may eventually cause correct-path misses. We call these misses *indirect misses*. For example, assume that: 1) Cache blocks *A*, *B*, *C*, and *D* all map to the same set, 2) We have a two-way associative set that contains blocks *A* and *B*, 3) *B* is the LRU block, and 4) *C* is the only cache block on the wrong-path. In this situation, the sequence of operations is as follows: Wrong-path block *C* replaces *B*, *D* replaces *A*, and, after a cache miss, *A* replaces *C*. If the wrong-path reference for block

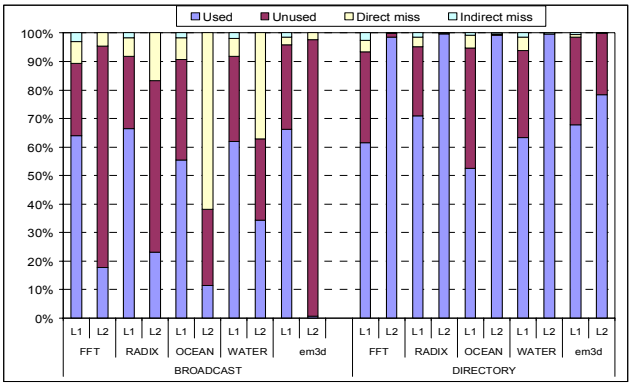

Figure 5. Replacements due to wrong-path memory references.

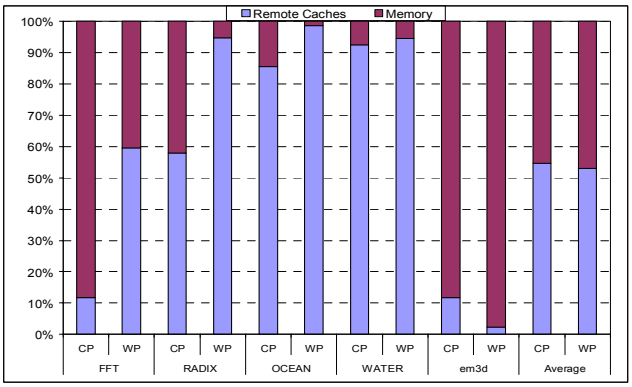

Figure 6. Servicing coherence transactions for broadcast-based SMPs for the correct-path (CP) and the wrong-path (WP).

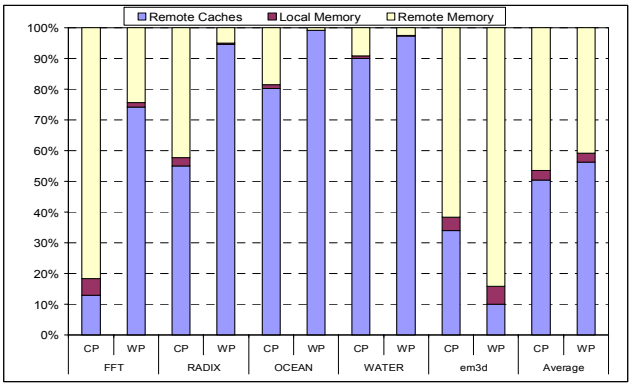

Figure 7. Servicing coherence transactions for directory-based SMPs for the correct-path (CP) and the wrong-path (WP).

C did not occur, then the correct-path reference for block *A* would have been a cache hit.

Figure 5 classifies the wrong-path-caused cache misses into the aforementioned four categories. The results show that 55% to 67% of the wrong-path replacements in the L1 data cache and 12% to 36% of the wrong-path replacements in the L2 are *used* in broadcast-based systems. *Direct misses* account for 5% to 62% of all wrong-path replacements and account for a higher percentage of wrong-path misses in broadcast-based SMP systems than for directory-based. Finally, *indirect misses* account for less than 5% of all wrong-path misses for most of the benchmarks and systems tested.

It is important to note that *direct* and *indirect* misses are responsible for the pollution caused by the wrong-path memory references. While they have similar effect on the L1 data cache for both broadcast and directory systems, their effects on L2 cache are different between the two SMP systems. For directory-based, almost all of the L2 replacements are *used*, while the opposite is true for broadcast-based. This suggests that wrong-path memory references have a greater effect on broadcast-based systems. However, a small number of remote misses caused by wrong-path loads may have a disproportionately large performance impact in a directory-based system, as compared to a broadcast-based system.

4.3 Servicing Cache Coherence Transactions

Broadcast-based cache coherence provides the lowest possible latency to retrieve data since misses can either be served by remote caches or shared memory. In contrast, in a directory-based SMP, misses can be served locally (including the local directory), at a remote home node, or by using both the home node and the remote node that is caching an exclusive copy, *i.e.*, a three-hop miss. The latter case has a higher cost because it requires interrogating both the home directory and a remote cache. Coherence misses account for most of the remote misses.

Figures 6 and 7 show how the correct-path and wrong-path cache coherence transactions are serviced for broadcast and directory-based SMP systems, respectively. The figures show that the results are similar for both SMP systems. Namely, remote caches service a greater percentage of the wrong-path misses than for correct-path misses for all benchmarks except *em3d*. For those benchmarks, the percentage of misses serviced by remote caches varies from 12% to 80% for correct-path loads and 55% to 96% for wrong-path loads. For the directory-based SMP, in all benchmarks, local memory services only a very small percentage of both correct-path and wrong-path memory references.

4.4 Replacements and Writebacks

As described in Section 2.2, wrong-path replacements may cause extra writebacks that would not occur otherwise. Figures 8 and 9 show the percentage increase in the number of replacements and writebacks due to wrong-path memory references. Figure 8 shows the percentage increase in the number of E (for directory MOESI) and S line replacements. E→I transitions – which increased by

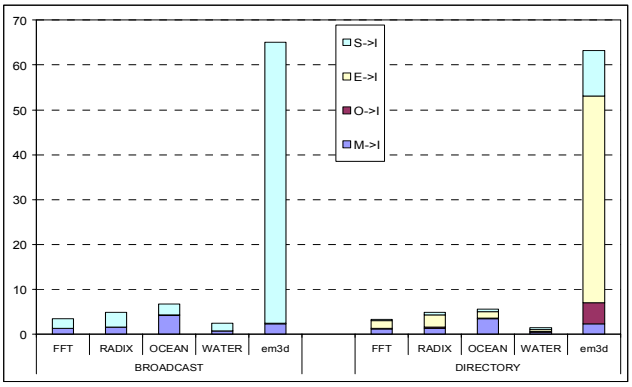

Figure 8. Percentage increase in the number of replacements due to wrong-path references in broadcast and directory-based SMPs.

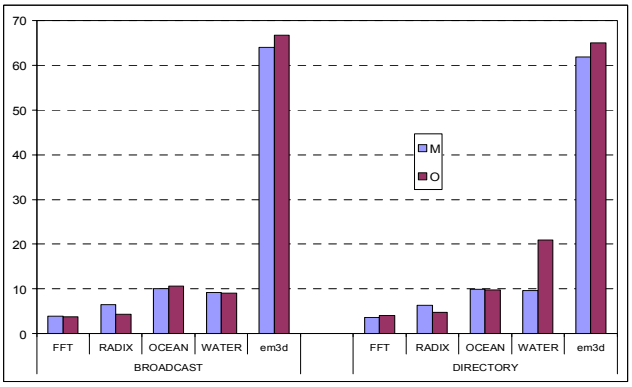

Figure 9. Percentage increase in the number of writebacks due to wrong-path references in broadcast and directory-based SMPs.

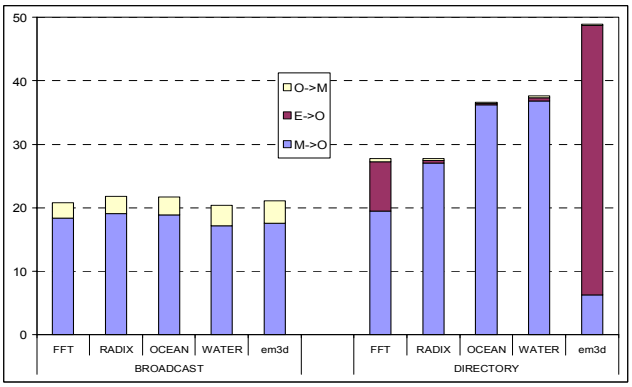

Figure 10. Percentage increase in the number cache line transitions for MOSI broadcast-based and MOESI directory-based SMPs

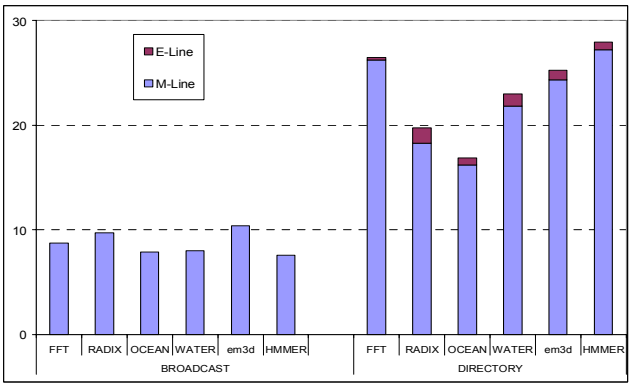

Figure 11. Increase in the write misses and extra invalidations due to wrong-path references in broadcast and directory-based SMPs.

2% to 63% – are particularly important since the processor loses the ownership of a block and, more importantly, the ability to silently upgrade its value, which can significantly increase the number of invalidations needed for write upgrades. For broadcast-based SMPs, the number of S line replacements account for a significant fraction of the total number of the replacements due to wrong-path references, while they are relatively insignificant for directory-based. For *em3d*, there is a large increase in both the replacements and writebacks.

Figure 9 shows that wrong-path memory accesses increase the number of writebacks from 4% to 67%. It is important to note that writebacks may result in additional stall cycles when an L2 cache miss occurs after the processor starts to perform a writeback, since it cannot begin to service the miss until the writeback completes.

4.5 Cache Line State Transitions

Figure 10 shows the impact that wrong-path memory references have on the number of cache line state transitions. The results show that the number of cache line state transitions increase by 20% to 24% for a broadcast-based SMPs and by 27% to 44% for directory-based. Although the percentage increase is smaller for the broadcast-based system, the number of cache line state transitions is much higher to begin with.

A processor loses ownership of an exclusive cache block (M or clean E) when another processor references it. In order to regain ownership, that processor has to first invalidate all other copies of that cache block, *i.e.,* S→I, for all other processors. Section 2.4 describes this situation in more detail. Figure 11 shows that there is 8% to 11% increase in the number of write misses – each of which subsequently causes an invalidation – for broadcast-based SMPs; this percentage is higher, 15% to 26%, for the directory-based SMPs.

5 Filtering and replacement policies for wrong-path memory references

In Section 4, we described the effects that wrong-path memory references can have on the memory subsystem behavior of broadcast and directory-based SMP systems. In this section, we evaluate two enhancements that try to

minimize the negative effects of wrong-path memory references, while retaining their positive effects (*i.e.*, prefetching), to improve the performance of an SMP system without significantly increasing the complexity of the memory subsystem.

5.1 Reducing Cache Pollution via Filtering

Based on the results in Section 4, we propose a filtering mechanism that reduces the cache pollution due by *direct* and *indirect miss* wrong-path references, and by evicting the *unused* wrong-path blocks early. We apply our filtering mechanism to the L2 cache due to the long latency of L2 instructions.

We base our filtering mechanism on the observation that if a speculatively-fetched cache block is not used while it resides in the L1 cache, then it is likely that that block will not be used at all or will not be used before being evicted from the L2 cache [13].

In this paper, we evaluate exclusive L1 and L2 caches. A block that misses both in L1 and L2 allocates a line only in the L1 cache. Then, when a block is evicted from the L1 cache, it is written to L2.

Our filtering mechanism works as follows: If a wrong-path block is evicted from the L1 cache before being used by a correct-path memory reference, it is allocated to the L2 cache only if its L2 set has an empty way, *i.e.*, at least one cache way is invalid. If not, then that cache block is discarded, *i.e.*, not allocated to the L2 cache, but written to memory only. A wrong-path block that services a correct-path reference is handled in the same way as a correct-path block.

We can further filter wrong-path blocks from being placed in L2 cache by canceling the wrong-path references in the L2 cache request queue as soon as the misprediction is known. For example, if a requested block is an L1 cache miss, a request is sent to the L2 cache controller and placed in a request queue. At the time that the L2 cache controller processes this request, if it is known that the load instruction was on a mispredicted branch path, then this request is simply discarded without being serviced. (If this request were not discarded, it would cause an L2 miss and could possibly replace a valid block in the L2 cache.) However, if there is an invalid line the set, the L2 cache controller services that wrong-path memory reference and overwrites the invalid line. Otherwise, the L2 cache controller processes this request as usual.

5.2 A Wrong-Path Aware Replacement Policy

Our second proposed enhancement is to make the cache replacement policy wrong-path aware. To accomplish this, when a block is brought into the cache, it is marked as being either on the correct-path or on the wrong-path. (There are several possible ways to design such a mechanism, but they are beyond the scope of this paper.) Later, when a block needs to be evicted from that set in the cache, assuming that all cache blocks are valid (if not, an invalid block is "replaced" first), wrong-path blocks are evicted first, on a LRU basis if there are multiple wrong-path blocks. However, a wrong-path block that services a correct path reference is marked as if it was on the correct-path, thus excluding it from the wrong-path replacement policy. If all cache blocks originated from a correct-path reference, then the LRU block in that set is chosen for eviction.

5.3 Performance Evaluation

Figure 12 shows the speedup results for the enhancements described in Sections 5.1 and 5.2. In this figure, there are a total of three enhancements: *replacement*, *filter*, and their combination, *i.e.*, *filter+replacement*.

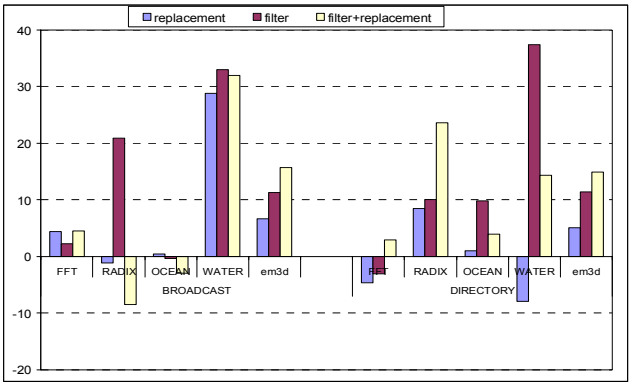

Figure 12. Percentage speedup in execution time for wrong-path aware replacement, L2 wrong-path filter, and for combination of both, *i.e.*, filter+replacement.

The results in Figure 12 show that a simple wrong-path aware replacement policy may perform very well for some benchmarks. For example, for *water*, all three enhancements yield speedups over 30% for the broadcast-based SMPs. Overall, the performance of the enhancements varies across benchmarks and systems. On average, filtering yields higher speedups than wrong-path aware replacement, while also outperforming replacement for all benchmarks for directory-based SMPs. For broadcast-based SMPs, filter performs better than replacement for *radix*, *water* and *em3d*. Employing a simple wrong-path replacement policy does not significantly improve the performance of *ocean* and *fft*.

6 Related work

To best of our knowledge, no previous work examined the effects that wrong-path memory references have on SMPs. However, several papers examined the effect that speculative execution had on the performance of

uniprocessor systems. Mutlu *et al.* [14, 15] analyzed the performance impact that wrong-path references have for different memory latencies and instruction window sizes. Their results showed that the major reason for performance degradation due to wrong-path memory references is L2 cache pollution.

Sendag *et al.* [19] proposed using the fully-associative *Wrong-Path Cache* (WPC) to eliminate the cache pollution caused by wrong-path references. The WPC stores data brought into the processor by wrong-path load instructions and evicted from the L1 cache. The processor accesses the WPC and the L1 data cache in parallel. Hence, the WPC functions both as a victim cache [8] and a buffer to store data fetched by wrong-path references. This approach eliminates the pollution caused by wrong-path references in the L1 cache. Sendag *et al.* [20] also studied the effects of incorrect speculation on the performance of a concurrent multithreaded architecture. They analyzed how wrongly-forked threads affected the memory system performance in addition to the known effects by the wrong-path load instructions in a uniprocessor.

Mutlu *et al.* proposed using the L1 caches as filters to reduce the pollution in the L2 cache caused by speculative memory references, including both wrong-path and prefetched references [13]. Their mechanism takes advantage of the observation that pollution in the L1 cache caused by speculative references has less impact on the performance than pollution in the L2 cache. Their approach reduces the L2 cache pollution due to speculative references for both for out-of-order and runahead processors, without requiring extra storage to hold the data fetched by the speculative references.

Finally, Pierce and Mudge studied the effect of wrong-path memory references on cache performance [16]. Their study used trace-driven simulation, where they injected a fixed number of instructions to emulate the wrong-path. However, this is not realistic because the number of instructions executed on the wrong-path is not fixed in a real processor [5]. They also introduced an instruction cache prefetching mechanism, which shows the usefulness of wrong-path memory references to the instruction cache [17]. Their mechanism fetches both the fall-through and target addresses of conditional branch instructions, *i.e.*, their mechanism prefetches instructions on the taken and not-taken paths.

It is important to note that all of the papers that were described in this section used simulation environments that did not run an operating system on top of the simulator, thus ignoring the effect of the operating system. In this paper, we used full-system cycle-accurate simulator (GEMS based on Simics), which accounts for operating system effects, and measures the wrong-path effects in both user and supervisor modes.

7 Conclusion and Future Work

In this paper, we evaluate the effects of executing wrong-path memory references on the memory behavior of cache coherent multiprocessor systems. Our evaluation reveals the following key conclusions:

1. It is important to model wrong-path memory references in cache coherent shared-memory multiprocessors. Neglecting to model them may result in incorrect design decisions, especially for future systems with longer memory interconnect latencies and processors with larger instruction windows.
2. For SMP systems, not only do the wrong-path memory references affect the performance of the individual processors due to prefetching and pollution, they also affect the performance of the entire system by increasing the number of cache coherence transactions, the number of cache line state transitions, the number of writebacks and invalidations due to wrong-path coherence transactions, and the amount of resource contention (buffer usage, bandwidth, *etc.*).
3. For a workload with many cache-to-cache transfers, wrong-path memory references can significantly affect the coherence actions.
4. Finally, simple mechanisms such as filtering unlikely to-be-used wrong-path blocks from being placed into L2 or making the replacement policy wrong-path aware can significantly improve the SMP performance.

Future Work*:* A thorough evaluation of wrong-path effects requires a detailed examination of how memory subsystem parameters (*e.g.*, cache size, associativity and block size), varying numbers of processors, different interconnections networks, and different multiprocessing models (*e.g.*, chip multiprocessor-based SMPs) change the effects that wrong-path references have on SMP performance.

Second, while we evaluated two possible enhancements in this paper, our primary goal was not to comprehensively study specific enhancements to improve the performance of SMP systems, but rather to study their potential effects. However, in our future work, we plan to develop more accurate filtering and replacement mechanisms to minimize the negative performance effects of speculative memory accesses.

Acknowledgments

We would like to thank Thomas Wenisch and Babak Falsafi for supplying us with the *em3d* benchmark.

References

[1] R. Bahar and G. Albera. Performance analysis of wrong-path data cache accesses. Workshop on Performance Analysis and its Impact on Design, 1998.

[2] R. Bhargava, L. John, and F. Matus. Accurately modeling speculative instruction fetching in trace-driven simulation. IEEE Performance, Computers and Communications Conference, 1999.

[3] P. Chang, E. Hao, and Y. Patt. Predicting indirect jumps using a target cache. International Symposium on Computer Architecture, 1997.

[4] Y. Chou, B. Fahs, and S. Abraham. Microarchitecture optimizations for exploiting memory-level parallelism. International Symposium on Computer Architecture, 2004.

[5] J. Combs, C. Combs, and J. Shen. Mispredicted path cache effects. Euro-Par, 1999.

[6] J. Dundas and T. Mudge. Improving data cache performance by pre-executing instructions under a cache miss. International Conference on Supercomputing, 1997.

[7] S. Iacobovici, L. Spracklen, S. Kadambi, Y. Chou, and S. G. Abraham. Effective stream-based and execution based data prefetching. International Conference on Supercomputing, 2004.

[8] N. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. International Symposium on Computer Architecture, 1990.

[9] D. Culler and J. Singh, Parallel Computer Architecture, Morgan Kaufmann, 1999.

[10] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. Computer Architecture News, Vol. 33, No. 4, September 2005

[11] C. Mauer, M. Hill, and D. Wood. Full-System Timing-First Simulation. Joint International Conference on Measurement and Modeling of Computer Systems, 2002.

[12] M. Moudgill, J. Wellman, and J. Moreno. An approach for quantifying the impact of not simulating mispredicted paths. Performance Analysis and Its Impact in Design, 1998.

[13] O. Mutlu, H. Kim, D. Armstrong, and Y. Patt. Cache filtering techniques to reduce the negative impact of useless speculative memory references on processor performance. Symposium on Computer Architecture and High Performance Computing, 2004.

[14] O. Mutlu, H. Kim, D. Armstrong, and Y. Patt. Understanding the effects of wrong-path memory references on processor performance. Workshop on Memory Performance Issues, 2004.

[15] O. Mutlu, J. Stark, C. Wilkerson, and Y. Patt. Runahead execution: An alternative to very large instruction windows for out-of-order processors. International Symposium on High-Performance Computer Architecture, 2003.

[16] J. Pierce and T. Mudge. The effect of speculative execution on cache performance. International Parallel Processing Symposium, 1994.

[17] J. Pierce and T. Mudge. Wrong-path instruction prefetching. International Symposium on Microarchitecture, 1996.

[18] E. Rotenberg, Q. Jacobson, and J. Smith. A study of control independence in superscalar processors. International Symposium on High Performance Computer Architecture, 1999.

[19] R. Sendag, D. Lilja, and S. Kunkel. Exploiting the prefetching effect provided by executing mispredicted load instructions. Euro-Par, 2002.

[20] R. Sendag, Y. Chen and D. Lilja. The Impact of Incorrectly Speculated Memory Operations in a Multithreaded Architecture. IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No. 3, pp. 271-285, March 2005.

[21] Y. Chen, R. Sendag, and D. Lilja. Using Incorrect Speculation to Prefetch Data in a Concurrent Multithreaded Processor. International Parallel and Distributed Processing Symposium, 2003.

[22] R. Sendag, Y. Chen and D. Lilja. The Effect of Executing Mispredicted Load Instructions on Speculative Multithreaded Architecture. Workshop on Multi-threaded Execution, Architecture and Compilation, 2002.

[23] A. Moshovos, G. Memik, B. Falsafi, and A. Choudhary, JETTY: Filtering Snoops for Reduced Energy Consumption in SMP Servers. International Symposium on High-Performance Computer Architecture, 2001.

[24] A. Moshovos. RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence. International Symposium on Computer Architecture, 2005.

[25] C. Saldanha and M. Lipasti. Power Efficient Cache Coherence. Workshop on Memory Performance Issues, 2001.

[26] M. Ekman, F. Dahlgren, and P. Stenström. TLB and Snoop Energy-Reduction using Virtual Caches in Low-Power Chip-Multiprocessors. International Symposium on Low-Power Electronics and Design, 2002.

[27] M. Ekman, F. Dahlgren, and P. Stenström: Evaluation of Snoop-Energy Reduction Techniques for Chip-Multiprocessors. Workshop on Duplicating, Deconstructing, and Debunking, 2002.

[28] J. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy. POWER4 system microarchitecture. IBM Technical White Paper, 2001.

[29] M. Wilkes. The memory gap and the future of high performance memories. Computer Architecture News, Vol. 29, No. 1, pages 2–7, March 2001.

[30] T. Yeh and Y. Patt. Alternative implementations of two-level adaptive branch prediction. International Symposium on Computer Architecture, 1992.

[31] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. International Symposium on Computer Architecture, 1995.

[32] G. Sohi, S. Breach and T. Vijaykumar. Multiscalar Processors, International Symposium on Computer Architecture, 1995.

[33] D. Culler, A. Dusseau, S. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. Parallel programming in Split-C. Supercomputing, 1993.

[34] A. Uht, D. Morano, A. Khalafi, and D. Kaeli. Levo - A Scalable Processor with High IPC. Journal of Instruction-Level Parallelism, Vol. 5, 2003.

[35] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. IEEE Computer, Vol. 35, No. 2, pages 50–58, February 2002.