Friendly Fire: Understanding the Effects of Multiprocessor Prefetches

Natalie D. Enright Jerger, Eric L. Hill, and Mikko H. Lipasti

Department of Electrical & Computer Engineering, University of Wisconsin-Madison
{enrightn,elhill,mikko}@ece.wisc.edu

Abstract

Modern processors attempt to overcome increasing memory latencies by anticipating future references and prefetching those blocks from memory. The behavior and possible negative side effects of prefetching schemes are fairly well understood for uniprocessor systems. However, in a multiprocessor system a prefetch can steal read and/or write permissions for shared blocks from other processors, leading to permission thrashing and overall performance degradation. In this paper, we present a taxonomy that classifies the effects of multiprocessor prefetches. We also present a characterization of the effects of four different hardware prefetching schemes--sequential prefetching, content-directed data prefetching, wrong path prefetching and exclusive prefetching--in a bus-based multiprocessor system. We show that accuracy and coverage are inadequate metrics for describing prefetching in a multiprocessor; rather, we also need to understand what fraction of prefetches interfere with remote processors. We present an upper bound on the performance of various prefetching algorithms if no harmful prefetches are issued, and suggest prefetch filtering schemes that can accomplish this goal.

1. Introduction

Dealing with the increasing relative latency of off-chip references to memory is one of the most pressing issues in processor design. One promising approach for mitigating this problem consists of anticipating future references and issuing speculative prefetches to move those blocks closer in the memory hierarchy. However, harmful effects due to limited cache capacity as well as contention for finite bus and memory bandwidth can erode or even overcome the benefits of prefetching. These are not fundamental obstacles, since capacity problems can be mitigated by adding a prefetch buffer, while more accurate prefetch algorithms can reduce the bandwidth overhead.

Both of these harmful effects also exist in multiprocessor prefetching; however, data sharing between processors introduces additional harmful side effects. In a single-writer invalidate protocol, a read prefetch can downgrade a remote line, causing it to lose write permission. If the remote processor has finished writing to this line, then no harm is done; however, if it needs to write the line again before the prefetched line is referenced, the prefetch has now slowed down the remote processor without accomplishing any useful work in the prefetching processor (illustrated in Figure 1). In the case of a write prefetch, the remote effects can be substantially more harmful. In the example in Figure 1, if the prefetch of A had been an exclusive prefetch, CPU 1 would have required additional coherence and data traffic to write the line a second time, since its copy of the line was invalidated.

	CPU 0		CPU 1		
Time	Instruction	Cache State	Instruction	Cache State	Coherence Traffic
T0			ST A	Modified	P1 Write Hit
T1	Prefetch A	Shared			P0 issues prefetch
T2				Owned	P1 copy downgraded
T3			ST A	Modified	P1 issues upgrade
T4		Invalid			P0 copy becomes invalidated

FIGURE 1. Harmful Prefetch Example.

Both hardware and software prefetching algorithms have been studied extensively in uniprocessors, and a variety of prefetching schemes have been both proposed and implemented. Prior work looks at both hardware [6], [7] and software [9], [10], [11], [14] prefetching in multiprocessor systems. Most of the evaluation has been done on scientific workloads [6], [7], [8], [9], [12], [23]. Coherence issues have been explored [6], [25], but an in depth analysis of the coherence issues associated with prefetching is absent from much of the prior work. Targeting only private data [26] is one technique to avoid the harmful effects of multiprocessor prefetching.

This work presents a novel taxonomy for classifying multiprocessor prefetches. We use this taxonomy to characterize a variety of prefetching algorithms on both commercial and scientific workloads. Another contribution not studied in prior work is a limit study of various prefetching algorithms on var-

ious benchmarks. Understanding and quantifying a prefetching algorithm's potential through a limit study can help tune the algorithm more than just the miss rate; miss rates alone are not adequate for understanding the performance effects in a modern system with out-of-order processors as they do not capture the harmful effects of additional upgrades. Our work shows that a substantial problem with MP prefetching is the introduction of additional communication traffic into the system that degrades remote processor performance. This problem can also be seen in prior work [26] by the increase in false sharing misses.

In our study of prefetching we name several orthogonal properties of harmful prefetches.

- Local conflicting prefetches: These exist in the uniprocessor case as well as in the multiprocessor case. A local conflicting prefetch is one that evicts a useful line from the cache, which is subsequently referenced before the prefetch is referenced.
- Remote harmful prefetches: A remote harmful prefetch is one that causes a downgrade in a remote processor followed by a subsequent upgrade in the same remote processor before the prefetch is either referenced by the prefetching processor or a demand reference occurs from another processor that causes the corresponding downgrade. From the prefetching processor's perspective this is a useless prefetch because it is never referenced before its eviction. These prefetches also have the potential to cause conflicts in the local cache. The ability to detect and eliminate a remote harmful conflicting prefetch is beneficial for both the local and the remote processor.
- Harmful speculation: Applies to our study of wrong path and exclusive prefetching. As the prefetches in wrong path prefetching are essentially speculative loads, a load that causes a remote harmful prefetch is considered potentially harmful speculation since, had that wrong path load not issued speculatively, the detrimental transitions in the remote processor would not have occurred. The same is true for exclusive prefetching which speculatively acquires exclusive access for a line before the store has reached the head of the reorder buffer.

An example of a remote harmful prefetch and a remote harmless prefetch are shown in Figures 1 and 2, respectively. Figure 3 illustrates the costly effect of a harmful exclusive prefetch. In this example, the exclusive prefetch invalidates line A in only CPU 1's cache, but the line could have been present and invalidated in multiple processors' caches, each of which could have subsequent references to A.

	CPU 0		CPU 1		
Time	Instruction	Cache State	Instruction	Cache State	Coherence Traffic
T0			ST A	Modified	P1 Write Hit
T1	Prefetch A	Shared			P0 issues prefetch
T2				Owned	P1 copy downgraded
T3	Read A	Shared			P0 Read Hit

FIGURE 2. Useful/Harmless Prefetch.

In Section 5, we present characterization data for these classes of harmful prefetches as well as for useful and useless prefetches. A useless prefetch is one that does not cause any detrimental transitions in a remote processor but is never used in the local processor. The only harmful side effect of such a prefetch is that it consumes additional bus bandwidth and resources without achieving any benefit for the system. In addition to quantifying each class of prefetch for different algorithms, we also correlate the harmful and useful prefetches to other events in the system. This correlation data can enable the construction of novel prefetching approaches and filters which avoid or eliminate harmful side effects and improve performance.

	CPU 0		CPU 1		
Time	Instruction	Cache State	Instruction	Cache State	Coherence Traffic
T0			LD A	Exclusive	P1 Read Hit
T1	Excl Prefetch A	Exclusive			P0 issues Excl Prefetch
T2				Invalid	P1 invalidated
T3			LD A		P1 Read Miss
T4		Owned		Shared	P1 Receives data from P0

FIGURE 3. Harmful Exclusive Prefetch Example.

The rest of this paper is organized as follows. Section 2 summarizes the original prefetch traffic and miss taxonomy presented in [19] and then expands it to include multiprocessor prefetches. Section 3 describes the prefetching algorithms examined by this work; Section 4 presents the simulation infrastructure followed by the results in Section 5. A summary of related work is presented next in Section 6 followed by our conclusions in Section 7.

2. Prefetch Taxonomy

Accuracy and coverage are two standard metrics used to evaluate the merits of a prefetching scheme. Accuracy measures the ratio of useful prefetches to the total number of prefetches issued. Coverage measures the reduction in demand cache misses due to the prefetching technique. In [19], Srinivasan et al. assert that prefetch accuracy and coverage are not sufficient metrics to evaluate a uniprocessor prefetching scheme because they give no indication of a prefetcher's effect on the other lines present in that cache set due to demand references. This taxonomy presents a comprehensive means to evaluate uniprocessor prefetching and can be extended to apply to multiprocessor prefetching schemes. In the next section, we will explain their taxonomy in more detail and then go on to present our novel expansion to that taxonomy in the following section.

2.1. Uniprocessor Prefetch Taxonomy

Srinivasan et al. [19] provides a new methodology and valuable insights into characterizing prefetching, but only considers these effects in uniprocessor systems. They present a taxonomy that incorporates the misses, both saved and induced by a prefetch and the additional traffic that may be induced by a prefetch. They note that it is important to consider how much extra traffic is added to the system as the result of prefetching.

The uniprocessor Prefetch Traffic and Miss Taxonomy (PTMT) breaks prefetches down into nine cases and classifies those nine cases as useful, useless, or polluting.

2.2. Multiprocessor Prefetch Taxonomy

Accuracy and coverage metrics do not account for the interactions between processors and do not address the effect prefetches have on communication misses. As mentioned previously, a remote harmful prefetch will cause additional coherence traffic. So in addition to the address bus and data bus traffic incurred by the useless prefetch, additional address bus traffic is incurred for harmful read prefetches, while additional traffic is required on both the address and data bus if the useless prefetch was an exclusive prefetch.

Using the groundwork laid by [19], we expand the uniprocessor PTMT to encompass multiprocessor interactions. There are many complex interactions that can occur when multiprocessor prefetches are introduced into the system but these interactions can be grouped into 6 broader categories of both coherence and data traffic which are used to simplify the collection and presentation of data. These 6 classifications are elaborated on in our full multiprocessor prefetch taxonomy, which is included in Appendix A. The state diagrams in Figures 4 and 5 show the different transitions a prefetch can make during its lifetime in the cache. Our taxonomy goes on to detail the ordering of various events that can cause these transitions. Figure 4 shows the data traffic component of a prefetch and Figure 5 depicts the coherence traffic component of a prefetch. For both figures, the transitions that are caused by exclusive prefetches are shown in parentheses. Please refer to Appendix A to understand what effect the initial coherence state of a remote block has on the classification of an exclusive prefetch. First, consider the transitions in Figure 4.

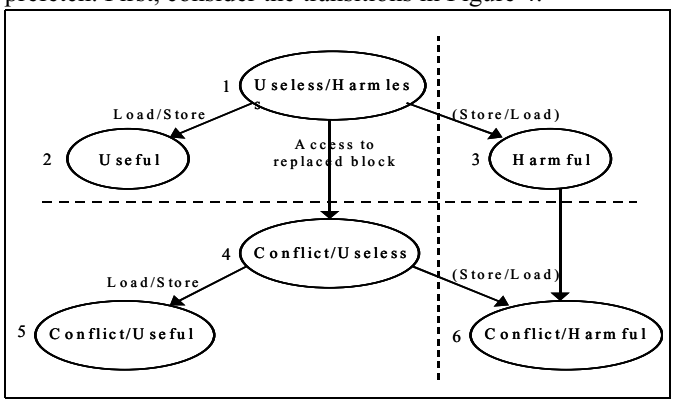

FIGURE 4. Harmfulness of data transactions.

- Useless/Harmless state (1): Entered when the local processor issues a prefetch. At this point, nothing is known about the future effects of this prefetch.
- From Useless/Harmless (1) to Useful (2): A demand reference will cause the prefetch to transition into this state.
- From Useless/Harmless (1) to Harmful (3): Since an exclusive prefetch also invalidates remote copies of the line, additional data traffic will be required to reinstall the line into the remote cache on a subsequent reference.

A prefetch can enter the Useful state (2) even if it caused a remote downgrade as long as the demand reference by the prefetching processor occurs prior to a subsequent upgrade by the remote processor. If the prefetch issued is a read prefetch, from the standpoint of additional remote data traffic, there are no harmful transitions. The lower two quadrants in Figure 4 track the side effects of a prefetch after it has been identified as a conflicting prefetch.

- From Useless/Harmless (1) to Conflict/Useless (4): A prefetch is identified as a conflict if the line is evicted from the cache is referenced before the prefetch is demand referenced.
- From Conflict/Useless (4) to Conflict/Useful (5): The prefetch is demand referenced after a demand reference has occurred to the conflicting line. The prefetch ends up netting zero savings in traffic or misses for the prefetching processor. This harmful side effect of an otherwise useful prefetch could be avoided through the use of a different replacement policy or a larger cache.
- From Conflict/Useless (4) to Conflict/Harmful (6): Same as the transition from state 1 to state 3 except a conflict was first detected in the local block.
- From Harmful (3) to Conflict/Harmful (6): Same as the transition from state 4 to 6 except the harmful transition was detected before the conflict. A remote load can only cause this transition if the prefetch in question was an exclusive prefetch.

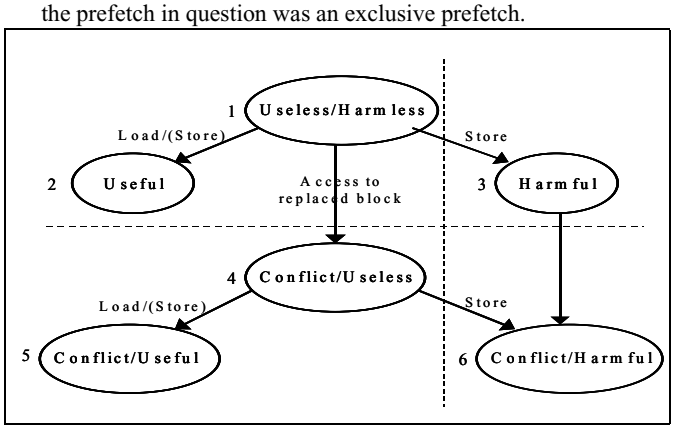

FIGURE 5. Harmfulness of address transactions.

Figure 5 shows the transitions prefetches can make with respect to coherence traffic. A read prefetch is considered useful from an coherence traffic perspective only for a demand load. For a write prefetch, a demand store will cause a transition from the Useless state (1) to the Useful state (2). From an coherence standpoint, read prefetches are not useful for stores but as shown above, they do save data traffic and only require an upgrade on a demand reference. Read prefetches steal exclusive ownership from a remote line which will cause a subsequent store to issue an upgrade request. This prefetch only induces an extra remote upgrade if the line was originally in the remote processor's cache in either the modified or exclusive state. A local prefetch that hits a remote processor's cache in shared, owned or invalid does not require an additional coherence request over the base case. Since write prefetches steal both ownership and data from a remote cache, any remote access to that line will be harmful. While a prefetch cannot transition from Harmful (3) to Useful (2), a percentage of harmful prefetches would have been useful had they occurred

later in execution; that is, after the remote processor's last access to the line. Multiprocessors introduce a new issue of timeliness in order for such prefetches to be useful.

In both figures, having a high number of prefetches that fall in the top left quadrant is desirable. Moving prefetches vertically from the bottom left quadrant to the top left quadrant can be achieved through increased cache capacity, the addition of a prefetch buffer, or by delaying the issuing of the prefetch until the conflicting references have completed. To move prefetches from the top right quadrant to the top left quadrant can be achieved by delaying the prefetch until the remote accesses have completed or by changing the coherence and synchronization mechanisms of the system. Filtering and prediction mechanisms can be used to reduce the number of prefetches in the right quadrants and will be discussed in section 5.

Multiprocessor prefetches have many of the same local effects as uniprocessor prefetches. Even though these effects are explained in the uniprocessor taxonomy [19], we include them to highlight specifically the ways prefetches can hurt performance for both prefetching and remote processors. Our taxonomy, in Appendix A, also indicates changes in remote traffic and misses due to prefetches. In uniprocessors, the timeliness of prefetches is important. A prefetch needs to occur far enough in advance that the data has arrived before the demand reference but not so far in advance that the prefetched line is evicted from the cache before the demand reference. A prefetch's ordering with respect to the last store by a remote processor directly impacts its harmfulness, adding an additional dimension of timeliness for prefetches. This timeliness with respect to remote prefetches is highlighted by the different prefetch outcomes of cases 4 (the prefetched line is locally referenced before a remote write) and 5 (the prefetched line is locally referenced after a remote write) of the taxonomy. If a prefetching algorithm exhibits a high percentage of prefetches in cases 5 and 10 (a prefetch that would have been useful except for an intervening remote write to that line) it will be evaluated to have low accuracy. However, in truth, the prefetching algorithm is accurately selecting addresses that will be demand referenced in the future but it not issuing them in a timely manner with respect to remote effects.

The three tables in Appendix A present the salient local and remote transitions that a prefetch can cause. With 29 enumerated cases, it is clear that prefetching in multiprocessor simulations introduces substantial new complexity and the potential to adversely affect performance. In Section 5, data will be presented that shows the relative number of these interactions to the number of prefetches issued by different algorithms.

3. Prefetching Algorithms

3.1. Sequential Prefetching

Sequential prefetching is one of the few hardware prefetching algorithms that has been studied in multiprocessor systems [7], [8], [17]. The algorithm modeled here approximates the Power4 sequential prefetcher [21]. This prefetcher detects a sequential stream of either ascending or descending addresses and begins prefetching the next 5 sequential addresses. Our simulations can prefetch for a maximum of 32 different address streams. When the prefetcher encounters a page boundary it stops prefetching to avoid incurring a TLB miss. Much of the prior work that studied sequential prefetching focused on scientific applications, whereas this work presents a combination of commercial and scientific workloads. Sequential prefetching is also included because its behavior in uniprocessor systems is well understood and it has been implemented in real systems.

3.2. Content Directed Data Prefetching

Content Directed Data Prefetching (CDDP) was introduced by Cooksey et al [5]. This prefetching algorithm uses a pointer matching heuristic to prefetch possible pointers into the cache. When a line is brought into the cache, it is scanned for possible pointers which are then prefetched. This algorithm is especially useful in applications that do not exhibit the regular access patterns that can be targeted with sequential or stride prefetching and that are pointer intensive. Our results show that certain commercial applications do not benefit from sequential prefetching because they lack regularity in their access patterns, so content directed prefetching can improve the performance of those workloads when absent of harmful effects. CDDP also has the advantage of targeting cold misses as it does not require any history to be established before it can begin prefetching.

3.3. Wrong Path Prefetching

Wrong path prefetching looks at the effect of fetching loads into the cache that are later determined to be on the wrong path following a branch. Work has been done to look at the effects of wrong path instruction prefetching in uniprocessors [18]. Recent work has been done to further understand the effect of wrong path data references as well as the ability to correlate these references to other system events [2], [16], however this work is limited to wrong paths effects in a single processor. A wrong path prefetch might fetch a cache line that is subsequently referenced by a right path memory operation, consequently reducing the stall time associated with that memory operation. However, like other kinds of prefetching, this wrong path prefetch might pollute the cache, evicting a potentially useful line, or it might downgrade a remote line causing a subsequent upgrade that slows the performance of the remote processor. Studying this scheme gives insight into when and why the speculation of loads can be advantageous or harmful. In contrast to sequential prefetching and CDDP, wrong path prefetching does not require any additional bandwidth over the base case as the base case permits these loads to execute speculatively.

3.4. Exclusive Prefetching

Exclusive prefetching allows the processor to speculatively issue a store to the memory system before it reaches the head of the reorder buffer. That store could be on the wrong path and therefore never executed which can result in harmful prefetches. In some cases, delaying the store until commit could have better performance than speculatively issuing the store depending on the impact the store has on the work being done by remote processors. We analyze exclusive prefetching in part because like sequential prefetching, it has been implemented in real systems and to highlight the differences between read and write prefetches.

4. Simulation Methodology

Our prefetching infrastructure is integrated into PHARMsim, a full system multiprocessor simulator [4], [13]. Our simulator is an aggressive out of order sequentially consistency implementation that models a near-RTL level MP system with a split transaction bus. The simulator implements the MOESI coherence protocol and sequential consistency with speculative loads.

One of the goals of this work is to present a limit study for several prefetching algorithms. Our prefetching infrastructure is able to mask the effects of cache pollution and remote effects but not the side effect of increased memory bandwidth requirements.

The mechanism used to eliminate the negative effects of cache pollution is an infinitely sized prefetch buffer that holds prefetches prior to being demand referenced. This approach effectively eliminates the impact of cache pollution as useless prefetches will never enter the cache, but it does nothing to eliminate remote effects, i.e. downgrades induced by prefetches. To solve this problem, we track prefetches in the remote cache which allows us to undo harmful state transitions caused by prefetches. In the situation of a prefetch necessitating the downgrade of an exclusive or modified line to owned, we record the necessary downgrade in the remote cache. If the next reference is upgrade by the owning processor we invalidate the prefetch and upgrade the line without an explicit upgrade transaction (zero latency). We classify this prefetch as harmful as it would have added additional coherence traffic for the upgrade without the prefetch being useful. However, if the next reference is a hit to the prefetched line, we then officially downgrade the remote line to owned and call that a harmless/useful prefetch since a demand reference would have caused the same downgrade.

We also use this infrastructure to quantify prefetches that would be detrimental to the prefetching processor due to cache conflicts or pollution. When a prefetched line is entered into the prefetch buffer, we mark the cache line which would have been evicted according to the LRU list for the set. This information makes it possible to identify the prefetch as a conflict if the local cache line is subsequently accessed by the processor prior to the prefetch being referenced. The prefetch remains in the prefetch buffer but is marked as a conflict prefetch so on a demand reference it can be properly classified as a conflict/useful prefetch. Harmful effects of a prefetched block are also tracked after the prefetch block has been marked as a conflicting prefetch. Conflict issues are difficult to track precisely because the introduction of a miss or a prefetch will change the LRU stack. Also, a subsequent demand miss might come along and invalidate the conflicting block associated with the prefetch. Useless prefetches have a potential side effect of writing a modified block back to memory sooner than the demand case, speeding up subsequent remote references to the written back block. As we hold useless prefetches in a separate buffer we do not cause these early write backs seen in a naive implementation.

Results are presented for a variety of workloads running in a 4 processor configuration. The applications include Barnes-Hut and Ocean from the Splash2 suite [3], [28], SPECjbb2000 [20], TPC-B and TPC-H [22]. The simulation parameters are presented in Table 1. The descriptions of the benchmarks run and their baseline performance are given in Table 2.

Table 1: Simulation Parameters

Processor Parameters	
decode/issue/commit	8/8/8
RUU/LSQ size	128/64
Functional Units	8 Int ALUs, 3 Int Mult/Div, 3 FP ALUs, 4 FP Mult/Div 3 LD/ST Ports
Branch Predictor	Combined bimodal (8k entry)/gshare (8k entry) with 8k choice predictor, 8k 4-way SA BTB, 64 entry RAS
Memory System	
L1 I Cache (latency)	128K 2 way set associative (1 cycle)
L1 DCache (latency)	128K 2 way set associative (1 cycle)
L2 Unified Cache	4MB 8 way set associative (12 cycles)
blocksize (all caches)	64 bytes
Data network latency	400 cycles
Address network latency	50 cycles

5. Simulation Results

5.1. Prefetching Performance

Our assertion is that naively implementing a variety of prefetching algorithms in multiprocessor systems can degrade performance. Table 3 shows the number of prefetches per thousand memory references for each algorithm and benchmark. Data showing the impact these four prefetching schemes

Table 2: Benchmark Descriptions

Benchmark	Description	Instr executed	IPC 4 proc
SPECjbb	Standard java server workload utilizing 4 warehouses executing 6400 requests	~1.7B	4.95
TPC-B	Standard OLTP benchmark with 20 clients executing a total of 1000 transactions against an in-memory DB2 v6.1 relational database	~530M	1.61
TPC-H	Transaction Processing Council's Decision Support IBM DB2 v6.1 running query 12 on 512MB database	~2.2B	1.85
Barnes-Hut	8K particles, full end-to-end run including initialization	~1.9B	7.13
Ocean	258x258 full end-to-end run with initialization	~1.2 B	5.17

can have on performance is presented in Figure 6. For each benchmark, Figure 6 shows the cycle counts normalized to a baseline simulation with no prefetching for each of the three prefetching schemes (CDDP, Sequential, and Exclusive) both with and without our idealized prefetch buffer which filters out harmful prefetches. Also shown is the performance of a scheme where all harmful wrong-path prefetches are removed (Wrong Path). We use statistical simulation methods [1].

Table 3: Prefetches/Thousand L2 References

Benchmarks	CDDP	Seq	Wrong Path	Excl.
SPECjbb	91	2	5	52
TPC-B	95	5	19	0.54
TPC-H	207	41	3	151
Barnes-Hut	97	6	8	0.43
Ocean	4.7	26	.1	13

CDDP is the most aggressive prefetching algorithm for the commercial workloads. As seen in Figure 6, the naive implementation of CDDP results in performance degradation. However, by eliminating the harmful effects of CDDP we see performance improvement. Sequential prefetching shows modest improvements across all benchmarks and significant performance improvement for ocean. The irregular access patterns of commercial workloads prevent sequential prefetching from improving performance significantly.

Some of our results show an increase in cycle count for wrong path prefetching. Naive wrong path prefetching is equivalent to the base case and has a normalized cycle count of 1. Wrong path prefetching with the idealized prefetch buffer extends the lifetime of a cache line in the level 1 cache by holding speculative loads in the buffer until they commit. This increase in lifetime results in an increase in dirty misses; more often remote processors need to incur additional latency while the dirty block is flushed out of the level 1 cache which degrades performance. The addition of the idealized prefetch buffer also changes the LRU list causing the caches to no longer see the same reference stream as the baseline case. These changes in replacements in both levels of cache adversely affects the performance of the wrong path idealized case.

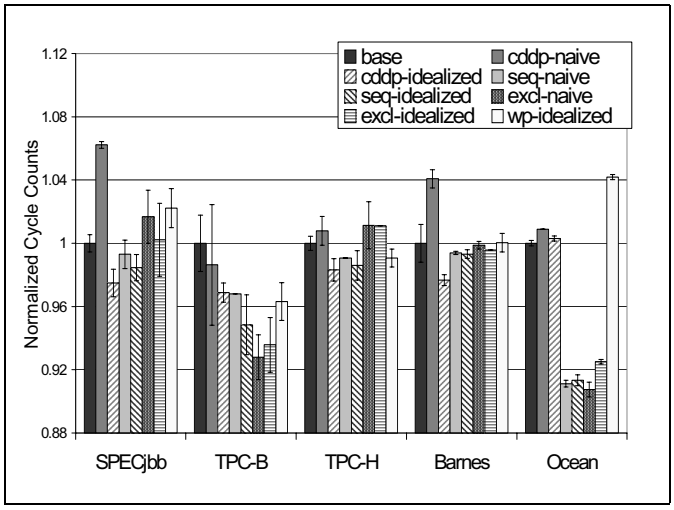

FIGURE 6. Performance with and w/out ideal prefetch buffer.

5.2. Breakdown of Prefetches

Figures 7 and 8 present the breakdown of prefetches according to our Multiprocessor Prefetch Traffic and Miss Taxonomy (MPTMT). The results indicate that a significant number of prefetches fall into the most detrimental state, conflict/harmful (as many as 35%). Useless prefetches also represent a significant percentage of prefetches issued. Eliminating useless prefetches would reduce contention for shared resources and help to further improve performance.

Figure 7 shows the breakdown of prefetches for the first three prefetching schemes according to the states shown in Figures 4 and 5. CDDP exhibits the highest percentage of useless prefetches. By culling out these prefetches, more benefit could potentially be reaped as there would be less contention for the bus and for memory. Detailed simulation results (not presented here) indicate that the average round trip latency of a load miss is increased by the substantial number of prefetches that CDDP injects into the system. Even though the prefetch buffer infrastructure does not remove contention side effects, performance improvement is still possible through CDDP if the harmful prefetches are removed. Overall, sequential prefetching issues the smallest percentage of harmful and harmful/conflict prefetches of the three algorithms and also has the highest percentage of purely useful prefetches. Even for large caches, conflict prefetches can be a significant fraction of the prefetches seen in multiprocessor simulations.

CDDP exhibits the greatest percentage of harmful and conflict harmful prefetches across all the benchmarks; however, the remaining prefetching algorithms issue enough harmful prefetches to give less than ideal prefetching performance. Eliminating harmful wrong path references improves performance. The harmful prefetches range between 5% and 80% for

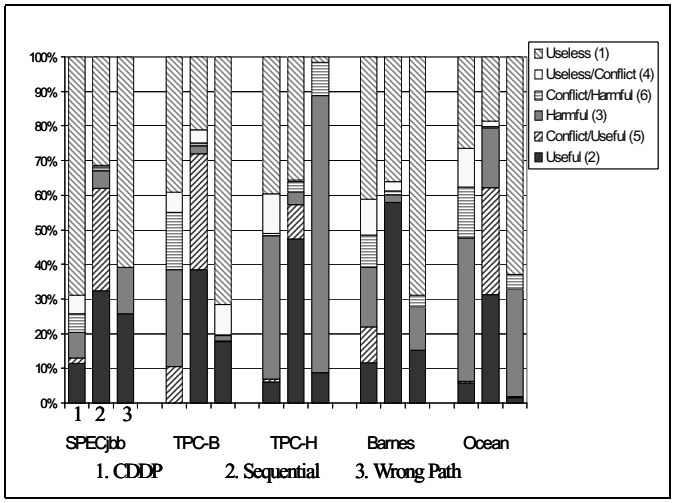

FIGURE 7. Read prefetch classification.

the benchmarks studied. Performance improvements are seen with both the naive and idealized sequential prefetcher. The improvements are greater with the latter. Eliminating harmful streams from the sequential prefetcher would free up those streams buffers to be used for additional useful prefetches. For CDDP, the commercial workloads exhibited the highest number of harmful prefetches. The irregular access patterns of commercial workloads result in a larger number of prefetches being issued which combined with more frequent data sharing results in a substantially higher number of harmful prefetches than is seen in the scientific workloads. In the case of CDDP, a significant percentage (as much as 83%) of harmful prefetches would have been useful to the prefetching processor had the prefetch occurred later in program execution. The other algorithms exhibit similar behavior but to a lesser extent than CDDP.

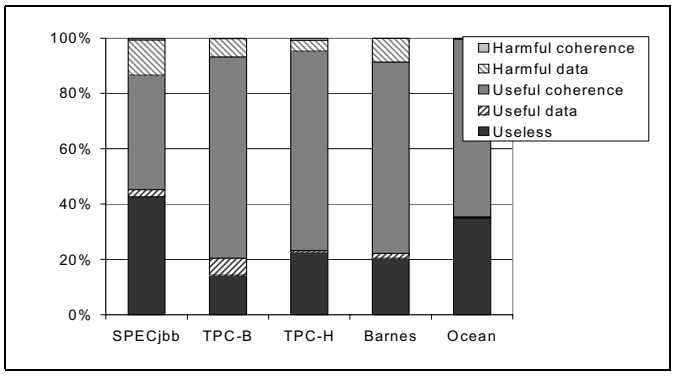

FIGURE 8. Exclusive prefetch classification.

Figure 8 breaks the harmful and useful prefetches down into coherence and data components for exclusive prefetching. As noted above, write prefetches have the effect of stealing both ownership and data, which adds a harmful data component that read prefetching does not possess. The harmful data traffic component of the bar implies that the remote cache had a valid copy and the next operation by that processor was a read. Harmful coherence traffic indicates that the remote processor had the line in an exclusive or modified state prior to the prefetch and the next access to that remote line is a write. With exclusive prefetching, it is interesting to note the breakdown of useful data/coherence versus harmful data/coherence. A significant amount of exclusive prefetching results in harmful data traffic which means that before the prefetch is referenced, a remote processor does a load of the data and finds its copy has been invalidated. The small percentage of harmful coherence traffic indicates that the remote processor will more likely read the line after the prefetch than write it.

Figure 9 shows the accuracy and coverage of the prefetching schemes across various benchmarks. For CDDP and sequential prefetching the coverage is less than 20% across all the benchmarks. Wrong path and exclusive prefetching show coverage ranging from 8.5% up to 26.3%, respectively. Exclusive prefetching has the highest accuracy of all the benchmarks, which is to be expected since it is issuing prefetches based on speculatively executed store instructions whereas CDDP and sequential prefetching make intelligent guesses at the addresses that are needed in the future. As previously noted, accuracy and coverage do not paint the complete picture of a prefetcher's effectiveness. For example, Barnes shows good coverage and accuracy for CDDP but its performance degrades by approximately 4% due to large number of harmful and conflict prefetches.

Our simulation infrastructure is able to mask all of the harmful effects of prefetching except for the increased bus and memory contention which could aggravate the latency of demand memory references. Due to this fact, our performance results show an approximate upper bound but as the increase in load latency is not significant in many cases, our upper bound is a valuable measurement. Contention caused by useless wrong path prefetches is not limited to bus bandwidth and memory contention. Useless wrong path prefetches also occupy space in the level 1 to level 2 request queues but do not consume space in the response queue which will lessen the delay of demand misses. By freeing up this queue space, right path loads will issue to memory and be returned from memory sooner. Alleviating pressure on these resources has potential for further performance improvement and finding a means to do so will be part of future work.

5.3. Identifying Harmful and Useful Prefetches

The MPTMT we have presented enables a detailed characterization of prefetching which can be extended to develop correlations between the different cases outlined above and system state and to exploit those correlations to filter out harmful prefetches. Among the correlations we looked at, two appear to have valuable potential as filtering mechanisms that can be applied to prefetching algorithms to filter out many of the harmful effects. These two correlations are prefetch correlations with Program Counter (PC) values and with the remote coherence state of the prefetched block. For prefetching algorithms that depend on the miss stream generated to issue additional prefetches, such as sequential and CDDP prefetching filtering prefetches out can potentially result in a dramatically

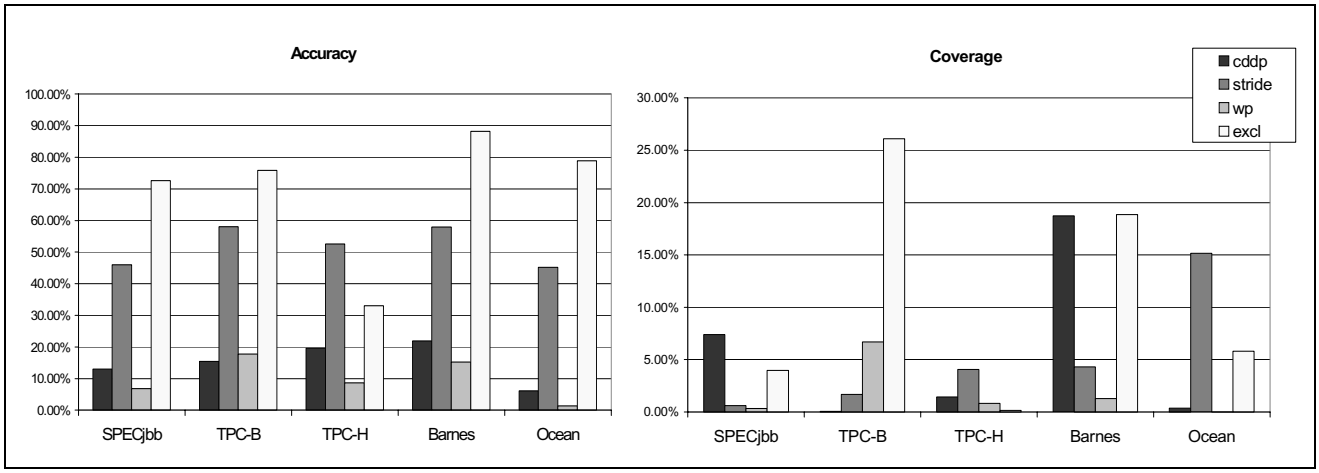

FIGURE 9. Accuracy and Coverage of Timely Prefetches.

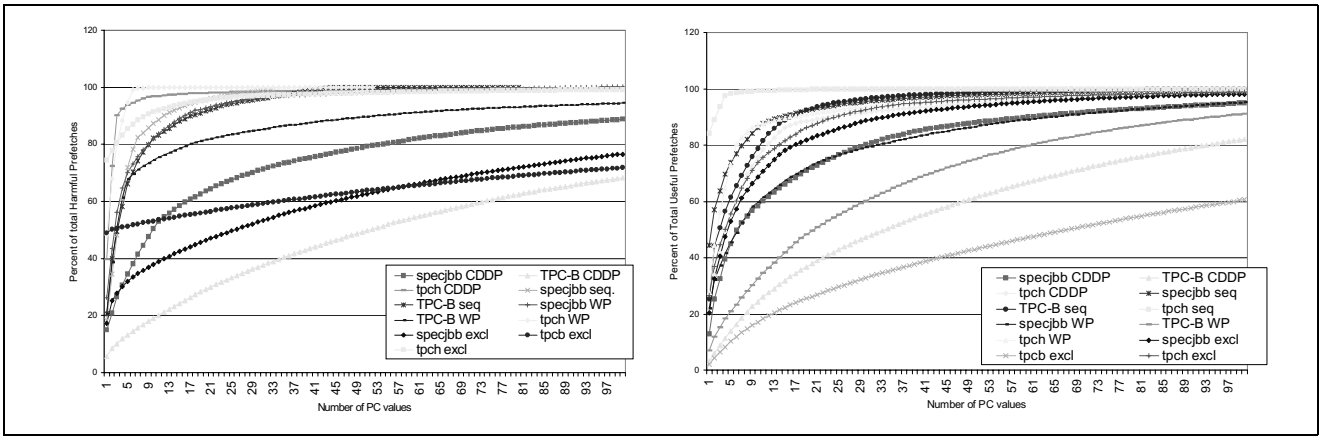

FIGURE 10. Harmful and Useful Prefetch Correlation to Program Counter values of prefetch initiating miss.

different set of prefetches than the original implementation making the potential improvement difficult to estimate.

Figure 10 gives the correlation of PC values to harmful and useful prefetches respectively for the commercial workloads studied. Only the commercial workloads are shown to prevent the graph from becoming too difficult to read. In our prefetching infrastructure we record the PC value of the demand miss that initiated a prefetching sequences and than record which PC values result in harmful or useful prefetches. With the exception of CDDP in SPECjbb and TPC-B, the majority of harmful prefetches correlate to only a few PC values. Approximately 15 PC values result in approximately 95% of the harmful prefetches. A very similar trend exists for useful prefetches. These trends are largely bimodal. The PCs that result in a large percentage of harmful prefetches result in few if any useful prefetches, and vice-versa.

We have implemented a small predictor table that increments a counter each time a prefetch is determined to be harmful. A small counter of only a few bits will suffice and PC values can be hashed to reduce to size of the table without sacrificing prediction accuracy. The PC based predictor requires a warm up period during which all prefetches are allowed to issue and their outcomes are recorded. After seeing 4 or more harmful prefetches that are associated with a single PC value and no intervening useful prefetches, future prefetches that correspond to that PC are inhibited. Using this scheme, we are able to achieve modest performance improvement with TPC-H for CDDP, sequential, and wrong path prefetching. While the performance gains seen in Figure 11 are not significant, the filter has successfully eliminated the harmful effects of these prefetching schemes. Implementing these prefetching algorithms can be desirable for single processor performance and with the addition of this predictor, the same algorithms will be performance neutral when put in a multiprocessor configuration. SPECjbb, TPC-H and Barnes are shown in Figure 11 as they suffered the most from harmful effects particularly with CDDP.

Harmful prefetches can also be correlated to the remote coherence state of the prefetched block. Prefetched blocks that are in a remote cache in either exclusive or modified state represent a large percentage of overall harmful prefetches, in some cases up to 99% of harmful prefetches. Rejecting prefetches that are in the exclusive or modified state will eliminate the majority of harmful prefetches for several algorithms. Unfortunately, this correlation is not as bimodal as the PC correlation. Rejecting these prefetches will result in rejecting some useful prefetches as well. For SPECjbb and TPC-H, rejecting prefetch requests to remote blocks in modified or

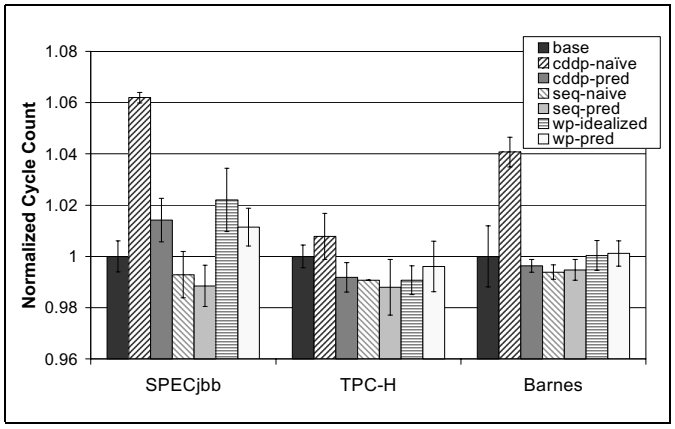

FIGURE 11. Performance of Program Counter Prediction Technique.

modified clean, will overcome much of the performance degradation seen in the CDDP naive case, however, there is no performance improvement over the baseline as a significant percentage of useful prefetch requests are also rejected.

Overall, the PC based predictor does a better job of eliminating harmful effects as it allows nearly all of the useful prefetches to still be issued. The PC base predictor inhibits prefetches locally, whereas in the coherence state filtering scheme, a prefetch request must still be sent out on the bus thereby consuming additional bandwidth and resources (MSHRs) that are left available for demand references in the PC based scheme.

6. Related Work

Some work has been done to study prefetching in multiprocessor systems. A significant body of work exists that looks at compiler based prefetching scheme for multiprocessors [15], [23], [24]. Other work has been done to devise hybrid software/hardware prefetching schemes [27]. We leave the study of software and hybrid schemes to future work. Our work differs from these studies in the workloads characterized as well as the classification of prefetches for different algorithms rather than focusing solely on performance and changes in a miss rates.

Work by Wallin et al. [26] explores the effect of one prefetching strategy on both scientific and commercial workloads. They draw two main conclusions in their work; first, that prefetching shared lines is harmful because it aggravates communication misses which are costly, and, second, that prefetching schemes that introduce large amounts of additional coherence traffic also degrade performance. They explore a method they call bundled capacity prefetching to reduce the amount of coherence traffic required for several cache lines while still prefetching a substantial amount of data. Our work supports part of their conclusion that prefetching shared cache lines can be bad, specifically the conclusion that prefetching frequently modified lines can be harmful.

Stride and sequential prefetching have been studied in a cache coherent NUMA architecture with a directory based protocol and an off chip second level cache [7]. This work used a different set of workloads and a different system configuration. One interesting conclusion they draw is that sequential prefetching performs equivalently if not better than stride prefetching in many cases and requires significantly less hardware overhead.

Garzaran et al. [8] present some characterization of load patterns for the SPLASH-2 benchmarks. According to their findings, these benchmarks are dominated by regular access patterns, namely scalar, sequential and stride patterns. These patterns are not nearly so prominent in commercial workloads necessitating further characterization to understand the best prefetching algorithm.

Tullsen et al. presents characterization data for compiler directed prefetching in shared memory multiprocessors running scientific applications [23]. With the rise of commercial workloads as a dominant application for multiprocessor systems, this work needs to be expanded to characterize the effects of prefetching on this class of applications. Our work is able to provide additional insight over this work as we use full system simulation where Tullsen et. al. used trace driven simulation. They emulate a software prefetching algorithm with perfect knowledge of non-shared data misses and simulate that. Our idealized study achieves similar goals for hardware based prefetching, only recording the effects of prefetches that are not harmful.

Neighborhood prefetching is a scheme that attempts to glean the benefits of sequential, stride and exclusive prefetching [11]. They do this by associating a neighborhood of lines with each miss and prefetching those lines. Lines to prefetch are also chosen based on the instruction that missed. Included in their table are two thresholds that are used to cull out harmful prefetches; however, they do not quantify the harmful prefetches. Their results reveal that while their prefetching scheme removes a substantial number of misses in all cases, the removal of those misses does not always translate into substantial performance improvement.

7. Conclusions

In this work, we have developed a multiprocessor prefetch traffic and miss taxonomy that builds on an existing uniprocessor taxonomy. Our taxonomy presents 29 interactions of prefetches with both the prefetching and remote processor that are of interest in evaluating a prefetch algorithms effectiveness. Only 9 interactions were necessary to develop a complete uniprocessor taxonomy [19]; this increase in interactions is indicative of the increase in complexity associated with multiprocessor prefetching. In addition to enumerating the 29 different interactions, we also present 2 state transition diagrams that summarize the characteristics of prefetches we are interested in. This means of classifying prefetches leads to a better understanding of the effects of prefetching on the remote processors in the system as well as on the prefetching processor.

Naive implementations of these prefetching algorithms

result in either insignificant performance improvement or degradation. Specifically, the potential for performance degradation makes implementing a prefetch algorithm in a multiprocessor system a risky proposition. However, this does not have to be the case. With increased knowledge and some additional complexity, these naive prefetching algorithms can be modified to yield significant performance improvement for a few benchmarks and eliminate the performance degradation and provide modest improvement for the remaining benchmarks. For future work, we plan to modify the prefetching algorithms studied here to achieve performance similar to the upper bound as well as study additional hardware and software algorithms that have been shown to be successful in uniprocessor simulations.

8. References

[1] Alaa R. Alameldeen and David A. Wood. Variability in architectural simulations of multi-threaded workloads. In *Proceedings of the 9th Annual International Symposium on High Performance Computer Architecture*, 2003.

[2] David N. Armstrong, Hyesoon Kim, Onur Mutlu, and Yale N. Patt. Wrong path events: Exploiting unusual or illegal program behavior for early misprediction detection and recovery. In *MICRO-37: Proceedings of the 37th International Symposium on Microarchitecture*, pages 119–128, 2004.

[3] Stanford SPLASH Benchmarks. SPLASH benchmarks. http://www-flash.stanford.edu/apps/SPLASH/.

[4] Harold Cain, Kevin Lepak, Brandon Schwarz, and Mikko H. Lipasti. Precise and accurate processor simulation. In *Workshop On Computer Architecture Evaluation using Commercial Workloads*, February 2002.

[5] Robert Cooksey, Stephan Jourdan, and Dirk Grunwald. A stateless, content-directed data prefetching mechanism. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, pages 279–290, October 2002.

[6] Fredrik Dahlgren and Per Stenstrom. Sequential hardware prefetching in shared-memory multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 6(7):733–746, July 1995.

[7] Fredrik Dahlgren and Per Stenstrom. Evaluation of hardware-based stride and sequential prefetching in shared-memory multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(4):385–398, April 1996.

[8] M.J. Garzaran, J.L. Briz, P.E. Ibanez, and V. Vinals. Hardware prefetching in bus-based multiprocessors: Pattern characterization and cost-effective hardware. In *Proceedings of Parallel and Distributed Processing 2001*, pages 345–354, February 2001.

[9] Edward H. Gornish, Elana D. Granston, and Alexandr V. Veidenbaum. Compiler-directed data prefetching in multiprocessors with memory hierarchies. In *Proceedings of the 4th international conference on Supercomputing*, pages 354–368, 1990.

[10] Alexander Klaiber and Henry M. Levy. An architecture for software-controlled data prefetching. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pages 43–53, 1991.

[11] David M. Koppelman. Neighborhood prefetching on multiprocessors using instruction history. In *IEEE PACT*, pages 123–132, 2000.

[12] D.A. Koufaty, X. Chen, D.K. Poulsen, and J. Torrellas. Data forwarding in scalable shared-memory multiprocessors. In *Proceedings of the 9th International Conference on Supercomputing*, pages 255–264, 1995.

[13] Kevin M. Lepak, Harold W. Cain, and Mikko H. Lipasti. Redeeming ipc as a performance metric for multithreaded programs. In *Proceeding of 12th International Conference on Parallel Architectures and Compilation Techniques*, pages 232–243, 2003.

[14] Chi-Keung Luk and Todd C. Mowry. Compiler-based prefetching for recursive data structures. *ACM SIGOPS Operating System Review*, 30(5):222–233, December 1996.

[15] Todd C. Mowry. Tolerating latency in multiprocessors through compiler-inserted prefetching. *ACM Transactions on Computer Systems*, 16(1):55–92, February 1998.

[16] Onur Mutlu, Hyesoon Kim, David N. Armstrong, and Yale N. Patt. Understanding the effects of wrong-path memory references on processor performance. In *Third Workshop on Memory Performance Issues*, 2004.

[17] Richard L. Oliver and Patricia J. Teller. Dynamic and adaptive cache prefetch policies. In *Proceeding of the IEEE International Performance, Computing and Communications Conference*, pages 509–515, February 2000.

[18] Jim Pierce and Trevor N. Mudge. Wrong-path instruction prefetching. In *International Symposium on Microarchitecture*, pages 165–175, 1996.

[19] Viji Srinivasan, Edward S. Davidson, and Gary S. Tyson. A prefetch taxonomy. *IEEE Transactions on Computers*, 53(2):126–140, February 2004.

[20] Systems Performance Evaluation Cooperative. SPEC benchmarks. http://www.spec.org.

[21] J.M. Tendler, J.S. Dodson, Jr J.S. Fields, H. Le, and B. Sinharoy. Power4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5–26, 2002.

[22] Transaction Processing Performance Council. TPC benchmarks. http://www.tpc.org.

[23] Dean M. Tullsen and Susan J. Eggers. Effective cache prefetching on bus-based multiprocessors. *ACM Transactions on Computer Systems*, 13(1):57–88, February 1995.

[24] D.M. Tullsen and S.J.Eggers. Limitations of cache prefetching on a bus-based multiprocessor. In *Proceedings of ISCA-20*, pages 278–288, New York, 1993.

[25] Steven P. Vanderwiel and David J. Lilja. Data prefetch mechanisms. *ACM Computing Surveys*, 32(2):174–199, June 2000.

[26] Dan Wallin and Erik Hagersten. Miss penalty reduction using bundling capacity prefetching in multiprocessors. In *Proceeding of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, April 2003.

[27] Zhenlin Wang, Doug Burger, Kathryn S. McKinley, Steven K. Reinhardt, and Charles C. Weems. Guided region prefetching: A cooperative hardware/software approach. In *Proceedings of ISCA-2003*, June 2003.

[28] S. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22th International Symposium on Computer Architecture*, June 1995.

Appendix A

The following tables present our expansion of the PTMT. The tables indicate which prefetches cause additional local traffic and misses in a format similar to the original PTMT. Each table describes the subsequent accesses to prefetched line A by the local processor and remote processors, and the subsequent access to line B which was replaced by the incoming line A. Each combination of actions has the corresponding state from the state transition diagrams shown above in Figures 4 and 5 in parentheses; where necessary, coherence and data state traffic are considered separately.

- Cases 2 and 3 show the difference between a write hit and a read hit on a read prefetched line. While the data transfer is useful, an additional coherence request is still required to gain write permission in case 2.
- Cases 4 and 5 highlight the notion that timeliness of a prefetch is important from a remote perspective. The timeliness of the local reference with respect to the remote reference determines whether a prefetch is harmful or useful. This notion of timeliness is not incorporated in the uniprocessor prefetch taxonomy.
- Cases 5 and 6 represent the purely harmful prefetches. These prefetches could later be locally referenced (resulting in a miss) or they could be useless.
- Case 7 represents conflict/useless prefetches. The additional local traffic caused by these prefetches is the same as presented in the PTMT [19].
- Case 8 represents conflict/useful prefetches. These prefetches do not introduce any remote harmful transitions but do require additional data and coherence traffic to reinstall line B in the cache.
- Case 9 and 10 are the most harmful of the read prefetches. These cases evict a useful line from the local cache and downgrades the remote line which subsequently sees a write.

Table 5 looks at the effects of write prefetches for the cases where the prefetched line is not present in the cache. In these cases write prefetches, acquire both the data and ownership of the line. Column 5 in Table 5 and columns 4 and 5 in Table 6 convey the change in coherence and data traffic and whether a miss is saved (-1), induced (1) or neither (0) due to the prefetch.

- Case 11 is useful because a write hit occurs in the prefetching processor before any reference to the line in the remote processor or before a conflicting reference in the local processor. Speculatively issuing this store was successful and should be allowed by any mechanism that attempts to filter harmful prefetches.
- Cases 12, 13, and 14 are harmful to the remote processor's performance because the line is invalidated before the remote processor is finished reading or writing it. Case 12 is labeled as data useful because had the write prefetch been converted to a read prefetch, it would have not caused any harm to the remote processor.
- Cases 17 and 18 are conflict harmful prefetches because they have invalidated the remote processor's copy of the line rather than just downgraded exclusive access. Case 17 does not induce additional coherence traffic because the remote line previously existed in the shared or owned state in the remote cache and would have required an upgrade request to write the line in the base case.

Table 6 also looks at the effects of write prefetches; however, for the cases presented in this table the prefetched line is already present in the prefetching cache in either the shared or the owned state. In this case, exclusive prefetching only prefetches ownership and not data. The column that considers the next reference to line B, the line replaced by prefetch A is removed from Table 6 because when the line is already present in the shared or owned state in the prefetching cache, the notion of conflicting references does not exist.

- Cases 19 and 20 represent harmless/useless prefetches. The first because there is no subsequent demand reference to the line and the second because all subsequent references are reads, rendering the upgrade transaction issued by the prefetch unnecessary.

Table 4: MPTMT Read Prefetches

Case	Prefetched block A in local cache	Reference to block B in local cache (replaced by A)	Remote access to A (in E or M state before prefetch)	Caused Additional Coherence Transaction	Classification
1	Replaced due to conflict/capacity	Replaced in base case	No reference or Read	no	Harmless/Useless(1)
2	Write Hit	Replaced or hit (later in logical time than demand reference to A)	No reference	no	Useful Data (2) Useless Address (1)
3	Read Hit	Replaced in base case	No reference or Read	no	Useful (2)
4	Read Hit	Replaced in base case	Write hit (ordered later than local reference to A)	no	Useful (2)
5	Hit (later in logical time than remote reference to A)	Replaced in base case	Write	Yes	Harmful(3)
6	Replaced due to conflict/capacity	Replaced in base case	Write	Yes	Harmful(3)
7	Replaced due to conflict/capacity	Miss (hit in base case)	No reference or Read	no	Conflict/Useless(4)
8	Hit (later in logical time than reference to B)	Miss (hit in base case)	No reference or Read	no	Conflict/useful(5)
9	Replaced due to conflict/capacity	Miss (hit in base case)	Write	Yes	Conflict/Harmful(6)
10	Hit (later in logical than remote reference to A)	Miss (hit in base case)	Write	Yes	Conflict/Harmful(6)

- Cases 21, 22, and 23 distinguish between data and coherence degrees of harmfulness. In these three cases, the coherence traffic is useless because either there is no subsequent reference or the reference is a read. The invalidation of data in the remote processor is harmful though because the remote processor subsequently reads the line again.
- Cases 24 and 25 are the two useful cases because the local processor sees its reference before the remote processor. Delaying the prefetch in case 27 would turn it into a prefetch of type 25.
- Cases 26, 27, 28 and 29 are harmful. Cases 28 and 29 are harmful because of the ordering of the remote write with respect to the local write. Delaying the write prefetch could prevent this harmful effect.

Table 5: MPTMT Write Prefetches. Remote interactions remote blocks in present in the remote cache and the block initially invalid in the local prefetching cache

Case	Next reference to Prefetch A in local cache	Next reference in local cache to block B (replaced by A)	Next remote access to A (valid before prefetch)	Remote Address/ Data/Miss	Classification
11	Write Hit	Replaced or hit (later in logical time than reference to A)	Read or Write Hit (later in logical time than local reference to A)	0/0/0	Useful (2)
12	Read Hit	Replaced or hit (later in logical time than reference to A)	Read Hit (later than local reference to a)	1/1/1	Useful Data (2) Harmful Address (3)
13	Hit (later in logical time than remote reference)	Replaced or hit (later in logical time than reference to A)	Read or Write (Remote cache in O or S prior to prefetch)	0/1/1	Useless Address (1) Harmful Data (3)
14	Hit (later in logical time than remote reference)	Replaced or hit (later in logical time than reference to A)	Read or Write (Remote cache in E or M prior to prefetch)	1/1/1	Harmful (3)
15	Never demand referenced	Miss (Hit in base case)	No reference	0/0/1	Conflict/Useless (4)
16	Hit (later in logical time than reference to B)	Miss (Hit in base case)	No reference	0/0/0	Conflict/Useful (5)
17	Never demand referenced	Miss (Hit in base case)	Write (Remote cache in O or S prior to prefetch)	0/1/1	Conflict/Harmful (6)
18	Never demand referenced	Miss (Hit in base case)	Read or Write (Remote cache in E or M prior to prefetch)	1/1/1	Conflict/Harmful (6)

Table 6: MPTMT Write Prefetches. The local cache already has a shared or owned copy at the time the prefetch is issued

Case	Next reference to Prefetch A (S or O state before prefetch) in local cache	Next remote access to A (valid before prefetch)	Remote Coherence/ Data/Miss	Local Coherence/ Data/Miss	Classification
19	No reference	No reference	0/0/0	1/0/0	Harmless/Useless (1)
20	Read	No reference	0/0/0	1/0/0	Harmless/Useless (1)
21	No reference	Read	1/1/1	1/0/0	Address Useless (1) Data Harmful (3)
22	Read	Read	1/1/1	1/0/0	Address Useless (1) Data Harmful (3)
23	Write (ordered later than remote reference to A)	Read	1/1/1	1/0/0	Address Useless (1) Data Harmful (3)
24	Write	Read (ordered later than local reference to A)	0/0/0	0/0/-1	Useful (2)
25	Write	Write (ordered later than local reference to A)	0/0/0	0/0/-1	Useful (2)
26	No reference	Write (remote cache in O or S prior to prefetch)	0/1/1	1/0/0	Harmful (3)
27	No reference	Write (remote cache in E or M prior to prefetch)	1/1/1	1/0/0	Harmful (3)
28	Write (ordered later than remote reference to A)	Write (remote cache in O or S prior to prefetch)	0/1/1	1/0/0	Harmful (3)
29	Write (ordered later than remote reference to A)	Write (remote cache in E or M prior to prefetch	1/1/1	1/0/0	Harmful (3)