Analyzing the Impact of Data Prefetching on Chip MultiProcessors

Naoto Fukumoto Tomonobu Mihara Koji Inoue Kazuaki Murakami
Department of Informatics, Kyushu University, Japan
{fukumoto, mihara}@c.csce.kyushu-u.ac.jp {inoue, murakami}@i.kyushu-u.ac.jp

Abstract

Data prefetching is a well known approach to compensating for poor memory performance, and has been employed in commercial processor chips. Although a number of prefetching techniques have so far been proposed, in many cases, they have assumed single-core architectures. In Chip MultiProcessor (or CMP) chips, there are some shared resources such as L2 caches, buses, and so on. Therefore, the effect of prefetching on CMP should be different from traditional single-core processors.

In this paper, we analyze the effect of prefetching on CMP performance. This paper first classifies the impact of prefetches issued during program execution. Then, we discuss quantitatively the effect of prefetching to memory performance. The experimental results show that the negative effect of invalidation of prefetched cache blocks is very small. In addition, it is observed that the current prefetch algorithms do not exploit effectively the feature of CMPs, i.e. cache-to-cache on-chip data transfer.

1 Introduction

Integrating multiple cores into a single chip, or Chip MultiProcessor (CMP), has emerged as a promising approach for microprocessor designs. This is because it achieves high-performance and low-power consumption simultaneously by means of executing multiple threads with a low clock frequency. Increasing the number of processor cores in a chip dramatically improves peak performance. However, since the shared resources such as I/O-pins and memory buses do not scale with the number of implemented cores, the memory-wall problem becomes more serious. To solve this issue, in CMP designs, it is very important to attempt reducing or hiding off-chip memory-access latency.

Data prefetching is a well known approach to improving memory performance. By fetching a cache block into an on-chip cache before it is demanded, we can hide off-chip access latency. However, we need to consider at least two negative effects caused by inappropriate prefetches, polluting effective cache capacity and dissipating shared memory buses, resulting in lower processor performance. Although a number of techniques for data prefetching have been proposed, they mainly focus on single-core processors. Of course, we can expect that the proposed prefetch schemes for single-core processors work well even in CMPs. However, the effects of data prefetching on CMPs may differ from that in single-core processors.

In order to propose more effective data prefetching schemes for CMPs, in this paper, we analyze the effects of data prefetching in detail. So far, some researchers have analyzed the impact of data prefetching on multiprocessor systems, in which processors are connected by an off-chip interconnection network. On the other hand, unlike the multiprocessor systems, CMPs can employ a fast on-chip network. This feature causes other effects of data prefetching on performance. Consequently, analyzing the behavior of prefetches on CMPs is worthwhile. The contributions of this paper are as follows.

- We propose an extended taxonomy for classifying prefetches on CMPs. In a prior research, a methodology to analyze the effects of data prefetching for multiprocessor systems has been proposed. We extend this approach to reflect the features of CMPs. Using this taxonomy, we can evaluate the effects of prefetch more precisely in CMP architectures.
- We analyze the impact of two prefetching algorithms quantitatively. All of the issued prefetches during program execution are classified based on the proposed taxonomy. From the experimental results, it is observed that the negative effect of invalidations of prefetched blocks is very small. Enright et al. [3] have reported that about 80% of prefetched blocks are invalidated in maximum on multiprocessor systems, whereas we show that this rate in CMPs is only 1%.
- In addition, from our experimental results, it is observed that the current prefetch algorithms do not exploit effectively the feature of CMPs, i.e. cache-to-cache on-chip data transfer. Only 5% of prefetches contribute to improving the memory performance for

978-1-4244-2683-6/08/$25.00 ©2008 IEEE

other on-chip cores. Thus, we conclude that it is strongly required to consider CMP-aware prefetchers.

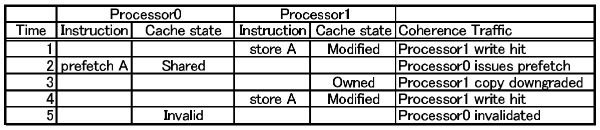
The rest of this paper is organized as follows. Section 2 summarizes the conventional method to analyze effects of prefetching on multiprocessor system (MPTMT [3]). In Section 3, we expand MPTMT to include CMP prefetching. Section 4 analyzes quantitatively the effects of prefetching on CMP. Section 5 concludes the paper.

2 Prefetch Taxonomy for MultiProcessors

Enright et al. presented MultiProcessor Prefetch Traffic and Miss Taxonomy (MPTMT) [3], which is an extend version of uniprocessor Prefetch Traffic and Miss Taxonomy (PTMT) [5]. Their motivation is that the conventional method is not sufficient in multiprocessor systems because of the lack of consideration for inter-processor effects.

In a multiprocessor system, we need to consider the following events relating to each prefetched block.

- Event 1: The prefetched cache block is referenced by the local processor.
- Event 2: A processor accesses to a block, which has evicted from the cache by the prefetched block.
- Event 3: The prefetch causes a downgrade followed by a subsequent upgrade in a remote processor. If this event occurs, the prefetched block is invalidated, and an additional invalidation request occurs. Figure 1 gives a concrete example, the occurrence of invalidation request due to issuing a prefetch. At Time 2, Processor0 prefetches a block from Processor1's cache, so that the block is shared. At Time 4, Processor1 updates the shared block, thus Processor1 issues a request to invalidate Processor0's copy. Note that this invalidation does not take place if Processor0 has not issued the prefetch for the shared block.



	Processor0		Processor1		
Time	Instruction	Cache state	Instruction	Cache state	Coherence Traffic
1			store A	Modified	Processor1 write hit
2	prefetch A	Shared			Processor0 issues prefetch
3				Owned	Processor1 copy downgraded
4			store A	Modified	Processor1 write hit
5		Invalid			Processor0 invalidated

Figure 1. Example of Additional Invalidation Request

In MPTMT, six states of a prefetched block are defined, and state transitions are performed based on these events during a lifetime of the prefetched block in a cache as shown in Figure 2. These states defined in MPTMT are as follows.

- Useless: This is the initial state. Until some events occur, the state maintains to stay in Useless.

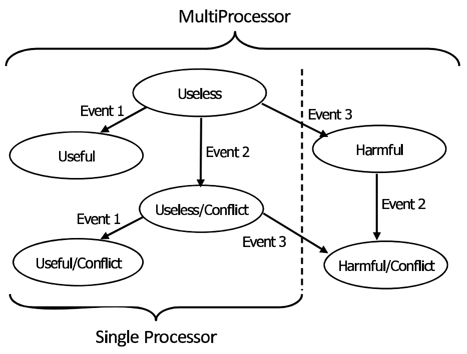


Figure 2. State Transition of Prefetched Block in Multiprocessor

- Useful: This state represents that the prefetched block has been accessed by the local processor. Therefore, we can expect a performance improvement. When Event 1 appears, the state transits from Useless to Useful.

- Useless/Conflict: This state shows that the prefetch has evicted an effective block from the cache. Consequently, memory performance is degraded due to the effect of cache pollution. When Event 2 occurs in Useless, the state moves into Useless/Conflict.

- Useful/Conflict: This is the same as Useless/Conflict except that the prefetched block has been accessed by the local processor. Therefore, the negative impact of cache pollution can be canceled by the prefetching effect. When Event 1 takes place in Useless/Conflict, the state transits to Useful/Conflict. However, the state does not transit from Useful to Useful/Conflict, when Event 2 occurs in Useful. This is because the cache pollution occurs in spite of not issuing prefetch, i.e. this pollution is originally caused by the memory reference from the local processor.

- Harmful: This state represents that prefetching has caused an additional invalidation. Since the invalidation process requires a broadcast transaction, traffic on the shared bus is increased. No performance improvement can be achieved in this case. The state transits from Useless to Harmful when Event 3 occurs.

- Harmful/Conflict: This is the same as Harmful except that the prefetching has evicted an effective block from the cache. In addition to an increase in the memory-bus traffic, the number of cache misses is also increased. This is the worst case. When Event 2 occurs in Harmful, or when Event 3 appears in Useless/Conflict, the state moves into Harmful/Conflict.

These states represent the impact of prefetching on memory performance. By means of counting the final state, which represents accumulate prefetch effects, for all prefetched blocks, we can analyze the effects of prefetching in detail.

In Figure 2, four states of the left side exist in both single processors and multiprocessors, but two states of the right side are only for multiprocessors. In multiprocessors, prefetching effects become increasingly more complex due to these two states.

3 Prefetch Taxonomy for CMPs

In this section, we extend the state transition diagram explained in Section 2 in order to reflect the features of CMPs. Here, we call a processor core which issues a prefetch *local core*. On the other hand, other processor cores in the CMP are referred to as *remote cores*. One of the advantages of CMPs is that we can exploit the efficiency of cache-to-cache data transfers. Even if the local core does not refer the prefetched block, it may be accessed by remote cores. In this case, the issued prefetch helps not the local core but the remote core, because access latency on cache-to-cache data transfers is much smaller than that on off-chip memory accesses. Therefore, in CMPs, we need to consider the following new event relating to a prefetched block.

- Event 4: The prefetched block loaded from lower level memory is referenced by a remote core.

Moreover, the following two states are defined in addition to the six states in multiprocessors , as shown in Figure 3.

- Useless/Remote: This state shows that the prefetched block has been accessed by a remote core before it is required by the local core. Therefore, the benefit of prefetching is provided not to the local core but to the remote core. When Event 4 occurs in Useless, the state transits to Useless/Remote.

- Useless/Conflict/Remote: This is the same as Useless/Remote except that the prefetching has evicted an effective block from the cache. Although the prefetching helps the remote core, the memory performance for the local core is degraded due to the effect of cache pollution. When Event 4 occurs in Useless/Conflict, or when Event 2 takes place in Useless/Remote, the state transit to Useless/Conflict/Remote.

In Table 1, we summarize the impact of prefetching on the three negative events relating to the memory performance. *Local L1 Miss Counts* are the number of L1 cache misses on the local core, and *Remote L1 Miss Counts* are that on remote cores. *Shared Bus Access Counts* represent the total number of bus-access events caused by original memory accesses, block prefetchings, and invalidations.

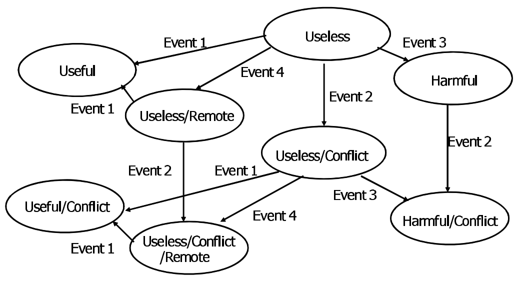


Figure 3. State Transition of a Prefetched block in CMP

Table 1. Impact of a Prefetch on local L1 Miss, Remote L1 Miss, Bus Access

State	Local L1 Miss Counts	Remote L1 Miss Counts	Shared Bus Access Counts (Data/Address)
Useless	± 0	± 0	+1/+1
Useless/Remote	± 0	-1	+1/+1
Useful	-1	± 0	± 0/± 0
Harmful	± 0	± 0	+1/+2
Useless/Conflict	$+1$	± 0	+2/+2
Useless/Conflict/Remote	$+1$	-1	+2/+2
Useful/Conflict	± 0	± 0	+1/+1
Harmful/Conflict	$+1$	± 0	+2/+3

The states including *Useful* decrease the number of local L1 cache misses, while the states having *Conflict* show an opposite feature. On the other hand, the prefetched block relating to *Remote* state decreases misses of remote L1 caches. For the traffic on the shared bus, only the state *Useful* does not give any negative effects. From these observations, we can understand that exploiting the Useless/Remote and Useless/Conflict/Remote states is a key for CMPs.

4 Data Prefetching Analysis on CMPs

4.1 Simulation Methodology

In order to support the prefetching analysis proposed in Section 3, we have modified M5 CMP simulation tools [2]. The CMP model evaluated in this paper is presented in Figure 4. Table 2 lists the configuration parameters used. We completely executed six benchmark programs selected from SPLASH2[6] with the input parameters listed in Table 3. For other parameters, we used default values. Our system sets the number of threads to be equal to the number of processor cores.

Our evaluation studies the performance impact of two prefetching algorithms; tagged prefetch[4] and stride prefetch[1]. The former scheme issues d of prefetches for block address $< a+1, a+2, \cdots, a+d >$ when an access to the address a misses the cache, or when a processor core accesses the address a block which has prefetched on

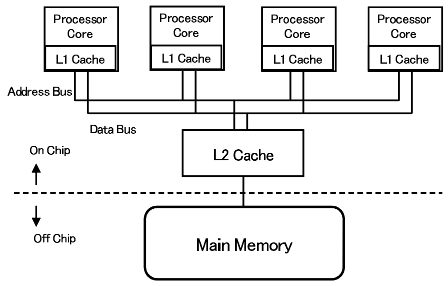


Figure 4. System Model

Table 2. Simulation Parameters

Processor	
The Number of Cores	4
Processor Model	Simple In-order Core
On-chip Caches	
L1 Instruction Cache	64KB, 2-way, 64B lines, 1 clock cycle
L1 Data Cache	64KB, 2-way, 64B lines, 1 clock cycle
L2 Shared Cache	4MB, 8-way, 64B lines, 12 clock cycles
Coherence Protocol	Write invalidate MOESI
Write Policy	Write back
Prefetch Mechanism	Prefetcher is attached to L1 cache. Prefetched block from main memory is loaded into L1 and L2 cache.
network	
On-chip bus width	64B
Off-chip bus width	16B
DRAM access latency	300 clock cycles

Table 3. The Input Parameters of Benchmarks

Benchmark	Input Parameter
Barnes	8k particles
FMM	16k particles
LU(contig)	512×512matrix
Radix	256K keys
Raytrace	teapot.env
Water(spatial)	512 molecules

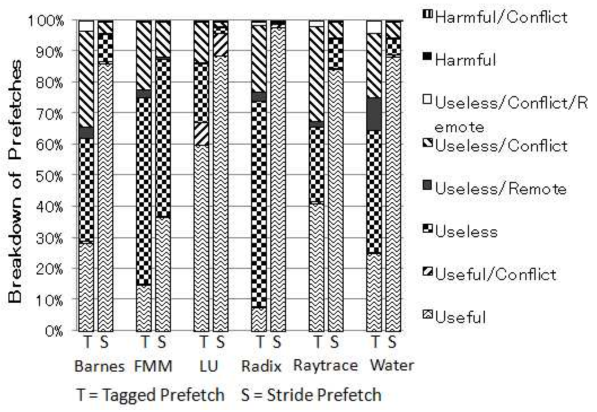


Figure 5. Classification of Prefetches Using Our Taxonomy

ahead. The parameter d is known as the degree of prefetching, which indicates the number of prefetches issued at a time. The latter scheme monitors memory access patterns to detect constant-stride array accesses originating from loop structures. This scheme attempts to detect a constant stride value for successive memory accesses. If a memory-access sequence accesses to $< a-2s, a-s, a >$, the prefetcher detects the value s as a stride. Then the blocks addressed by $< a+s, a+2s, \cdots, a+ds >$ are prefetched. The stride prefetch requires a reference prediction table (RPT) to memorize the history of memory access addresses. In this evaluation, we assume a 64-entry direct-mapped RPT. We use the constant value 5 as the degree of prefetching (d) for both prefetching algorithms.

4.2 Quantitative Analysis

4.2.1 Breakdown of Prefetches

Figure 5 presents the breakdown of prefetches based on our taxonomy proposed in Section 3. The y-axis is the breakdown of prefetches and the x-axis is the name of benchmark programs. Two bars represent the tagged prefetch and the stride prefetch from left to right.

One of the main features of CMPs is the cache-to-cache data transfers as explained in Section 3. Therefore, it is worthwhile to analyze the impact of prefetches which are categorized into Useless/Remote or Useless/Conflict/Remote. For the stride prefetch algorithm, the remote-access advantage does not appear for all of the benchmark programs. Although this effect can be seen for the tagged prefetch algorithm, its impact on the processor performance is very small. Actually, only 5% of the issued prefetches have the state of Useless/Remote or Useless/Conflict/Remote. Namely, these prefetch algorithms developed for single-core microprocessors do not exploit the feature of CMPs. These results suggest that we should develop CMP-aware prefetching algorithms.

Another discussion point is the negative impact of invalidation of prefetched blocks, i.e. the state of Harmful and Harmful/Conflict. From Figure 5, we can understand that almost all prefetches do not transit to these states. Consequently, in CMPs, this negative impact is negligible. The reason is discussed in Section 4.2.3.

Next, we consider the effectiveness of prefetches for local-cache accesses, i.e. the prefetches categorized into Useful or Useless states. For the stride prefetch algorithm, 80% of prefetches belong to the Useful state except for *FMM*. On the other hand, the tagged scheme issues many Useless prefetches which make negative impacts on a bus traffic and power consumption. The reason why the tagged approach does not work well comes from the lack of prefetching accuracy. In the tagged approach, a prefetch is issued not only on a cache miss but also when a prefetched block is referenced. This strategy tends to increase the number of prefetches. Consequently, if address prediction accuracy is not so high, prefetches having the state Useless takes place frequently.

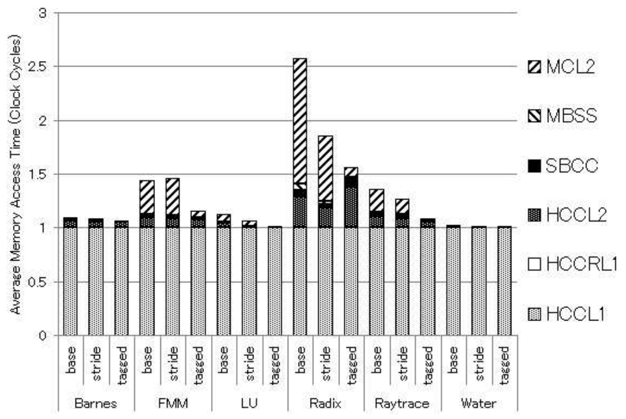


Figure 6. Average Memory Access Time

4.2.2 Average Memory Access Time

Although we have discussed the category of prefetches in Section 4.2.1, it is still not clear how much they affect to the memory performance. In order to clarify this point, in this section, we quantitatively analyze the impacts of prefetches on average memory access time (or *AMAT*). In our CMP model depicted in Figure 4, *AMAT* can be represented as follows.

$$\begin{aligned} AMAT &= HCC_{L1} + MR_{L1} \times \\ &\quad \{(1 - MR_{L1R}) \times (SBCC + HCC_{L1}) + \\ &\quad MR_{L1R} \times ((HCC_{L2} + SBCC) + \\ &\quad MR_{L2} \times (MBCC + MC_{L2}))\} \end{aligned} \quad (1)$$

- HCC_{L1}: L1 cache hit time
- MR_{L1}: Local L1 cache miss rate
- $SBCC$: Average shared bus access time (between L1 and L2 cache)
- MR_{L1R}: Miss rate of remote L1 caches; i.e. the rate that remote L1 caches do not have the demanded data when local L1 cache miss occurs.
- HCC_{L2}: L2 cache hit time
- MR_{L2}: L2 cache miss rate
- $MBCC$: Average off-chip bus access time (between L2 cache and main memory)
- MC_{L2}: Main memory access time

Figure 6 shows average memory access time to be calculated based on above equation. The y-axis is *AMAT* and the x-axis is the benchmark programs. The *base* in this figure represents the model which does not support any hardware prefetches. HCCL1, HCCRL1, HCCL2, SBCC, MBSS and MCL2 represent the average access time spent in the L1 cache, remote L1 caches, the L2 cache, the shared bus, the memory bus and main memory, respectively. These items are calculated by following expressions. Rest of terms including MR_{L1} and MR_{L2} are obtained from simulation results.

- HCCRL1=$MR_{L1} \times (1 - MR_{L1R}) \times HCC_{L1}$
- HCCL2=$MR_{L1} \times MR_{L1R} \times HCC_{L2}$
- SBCC=$MR_{L1} \times SBCC$
- MBSS=$MR_{L1} \times MR_{L1R} \times MR_{L2} \times MBCC$
- MCL2=$MR_{L1} \times MR_{L1R} \times MR_{L2} \times MC_{L2}$

Average memory access time is improved due to reducing the access time of main memory and L2 cache for almost all benchmark programs. In particular, in the benchmark programs that have large proportion of Useful prefetches such as *LU*, average L2 cache access time (HCCL2) is dramatically reduced. On the other hand, in the benchmarks having many Useless/Conflict prefetches such as *Radix* with tagged prefetch, prefetches increase L2 cache access time. This performance degradation is not observed in *Barnes* and *FMM* since the case of evicting effective prefetched blocks is considered as Useless/Conflict or Useful/Conflict.

Another interesting observation is the effect of memory bus accesses. As explained in Table 1, all of the prefetch states except for Useful require extra bus traffics. In multiprocessor systems, it is one of the most important factors to consider the efficiency of prefetches. However, unlike the multiprocessor systems, this negative impact is trivial on CMPs. This comes from the fact that the cost to perform on-chip bus transactions is much smaller than that for off-chip bus accesses, i.e. we can exploit high on-chip bus bandwidth. Actually, although 70% of tagged prefetches (in average) have non-Useful states as showed in Figure 5, the effects of SBCC on *AMAT* is negligible. Off-chip bus accesses also have little influence on *AMAT* due to a small number of L2 cache misses per instruction.

It should be noted that the tagged prefetch scheme achieves higher performance compared to the stride approach in spite of the lower ratio of Useful prefetches, i.e. the stride prefetch scheme has high ratios for Useful prefetches but its *AMAT* is large. This is because the total number of prefetches issued during each program execution is not the same. As showed in Table 4, the tagged prefetch scheme aggressively issues prefetch operations. Therefore, the number of Useful prefetches (not the ratio) of the tagged prefetch scheme is larger than that of the stride approach, resulting in higher memory performance.

Table 4. The Number of Issued Prefetches per Thousand Memory Accesses

Benchmark	Stride Counts	tagged Counts
Barnes	0.14	33
FMM	1.5	31
LU(contig)	3.7	6.5
Radix	8.7	157
Raytrace	1.6	31
Water(spatial)	0.05	4.7

4.2.3 Comparison with Multiprocessors

Enright et al. evaluated the effects of data prefetching on multiprocessor systems based on MPTMT explained in Section 2[3]. Although it is not easy to directly compare our analysis results with those reported in [3], in this section, we discuss the difference for prefetch effects on shared cache CMPs and multi-processor systems.

We have found that a large difference for the percentage of Harmful and Harmful/Conflict even if the same benchmark program (*Barnes*) and the similar prefetch mechanism (Sequential prefetching) are assumed. For the multiprocessor system analyzed in paper [3], the amount of prefetches categorized into Harmful and Harmful/Conflict ranges between 3% and 80% (in average 23%). In contrast, for the CMP platform discussed in this paper, that value is only 1% or less (in average 0.1%).

The reason comes from the difference of the cache size where the prefetched blocks are loaded. Figure 7 shows the percentage of invalidated prefetch blocks on the CMP platform. In this figure, we can see that the rate of invalidation decreases if we employ the smaller size of cache. This is because prefetched blocks can easily be evicted from the small L1 cache, so that the possibility to be invalidated is reduced. Contrary, if we have a larger cache, the percentage of the invalidations increases. In many multiprocessor systems, as assumed in Enright's work, prefetched blocks are loaded into the L2 cache, and the coherence actions take place at this level. On the other hand, in CMPs having a share L2 cache, the targets of coherence actions are the L1 caches. Therefore, we conclude that the unnecessary invalidations of prefetched blocks are not a serious problem in CMPs, whereas they largely affect the performance of multiprocessor systems.

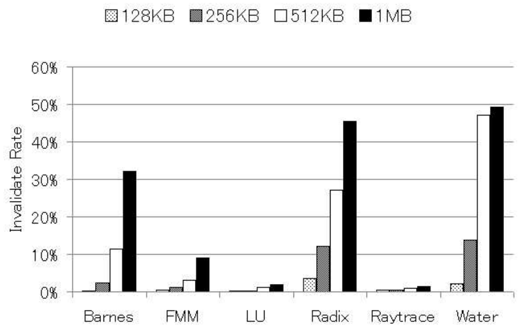


Figure 7. Invalidation of Prefetched Blocks

4.3 Sensitivity Analysis

4.3.1 Impact of the Number of Processor Cores

One of the most important design alternatives in CMPs is the number of processor cores. In this section, we discuss the sensitivity of prefetch effects to the number of cores. We assume that the evaluated CMP configuration consists of two, four, or eight processor cores. We use the same problem size for the execution of each benchmark program, and henceforth focus on the tagged prefetch algorithm which has achieved higher performance as showed in Section 4.2.2. Figure 8 shows the breakdown of prefetches for each CMP configuration.

Generally, increasing the number of cores makes total L1 cache capacity large because each core employs private caches. This side effect takes at least the following three characteristics.

- Reducing capacity cache misses: If we assume that the problem size of the target application program is a fixed value, the working set size for each processor core becomes small. Therefore, this positive effect improves the total cache miss rate.
- Increasing remote-cache hits: Since the total cache capacity becomes larger, the opportunity to take remote-cache hits is increased, reducing the total number of off-chip accesses.
- Increasing cache block invalidations: The number of cache blocks to be kept in coherent is increased. As a result, the block invalidation counts will be increased. This negative effect may worsen the bandwidth pressure.

Let us look at Useless/Remote state and Useless/Conflict/Remote state which represent the feature of CMPs, i.e. cache-to-cache block transfer. The rate of Useless/Remote state and Useless/Conflict/Remote state depends on the benchmark program. For instance, *FMM* and *Radix*, we see that the total ratio of these states tend to increase with the number of cores. This is because the second characteristic explained above, i.e. the increase in the remote-cache hits, appears clearly. On the other hand, for *Water*, the number of prefetches taking the advantage of cache-to-cache block transfer is reduced when the number of cores is increased. The reason is the reduction of working-set size as explained above. Since a number of memory accesses can be satisfied by the local L1 cache, the opportunity for accessing to remote L1 caches is decreased.

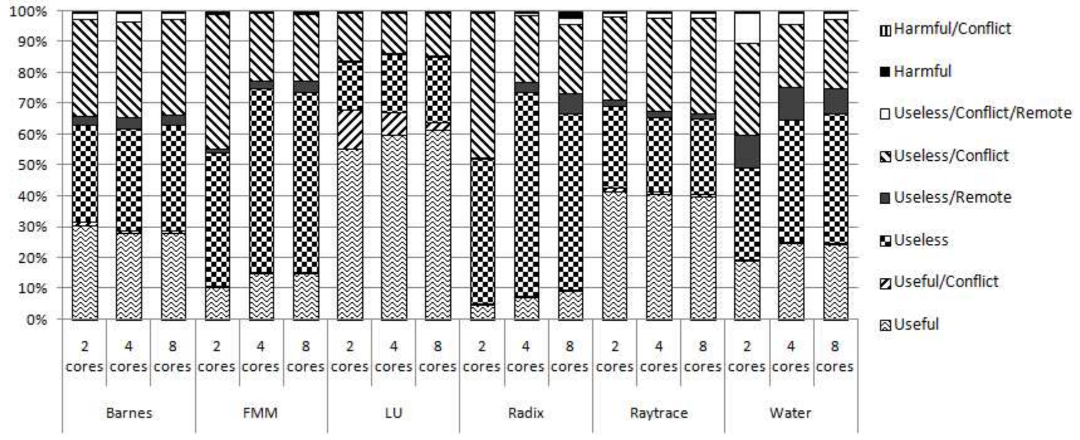


Figure 8. Breakdown of Prefetches with Tagged Prefetch versus the Number of Cores.

To summarize , these state is intricately changed with combination of the working set size and hardware configurations.

Next, we focus on Harmful and Harmful/Conflict prefetches. The rate of these prefetches is increased in all benchmark programs with increasing the number of processor cores, which result is caused by third characteristic as explained above. This result is less-visible due to tiny proportion of these prefetches in all benchmark programs except for *Radix*. However, in many number of cores system, invalidation of prefetched blocks effects may cause serious performance degradation.

4.3.2 Impact of L1 Cache Size

In our prefetch analysis, results strongly depend on how long each prefetched block resides in the cache. This means that the ability of the cache memory affects the efficiency of prefetches. One of the most important parameters for cache designs is the capacity (or cache size). L2 cache size change may make a bigger impacts on performance, but CMPs and single processors are similar in terms of effects of L2 cache size change because cache-to-cache data transfer or invalidation of prefetched blocks do not occur in the L2 cache. Consequently, in this section, the impact of the L1 cache size on the efficiency of hardware prefetches is discussed. Figure 9 presents the breakdown of prefetches with the varying L1 cache size from 32KB to 128KB.

We discuss the prefetches having the state Useless/Remote, which is a representative feature of CMPs. Generally, the time period in which prefetched block stays in the cache becomes longer with the increase in the cache capacity. Varying L1 cache size has a large impact on the lifetime of prefetched blocks. We can consider that increasing the cache size produces at least the following three effects.

- The opportunity to transit from the state Useless/Remote to Useful is increased due to the increased possibility to access to the prefetched blocks in the cache. In this case, both of the local and remote cores can take the contribution of the prefetch. *Barnes* and *LU* take this effects.

- The number of state transitions from Useless/Remote to Useless/Conflict/Remote is increased or decreased. This is caused by two factors. The increasing factor is that the possibility for taking the conflict events is also increased if prefetched blocks stay in the cache longer. This phenomenon is observed in *Water* from 64KB to 128KB. Another factor is occurring less conflict events by decreasing the rate of effective blocks to account for all cache blocks. This phenomenon is observed in *Water* from 32KB to 64KB.

- The number of prefetches which transits from Useless to Useless/Remote is increased or decreased. It depends on the balance between two factors: increasing factor or decreasing factor. First, as sum of the L1 caches capacity is larger, a processor core has more chances to access prefetched blocks in remote L1 caches. Second, since the hit rate of own L1 cache is improved, the opportunity of accesses to other L1 caches is decreased. These both effects can be seen in *Water*.

Next, we focus on the prefetch state Useless/Conflict/Remote. As explained above, some prefetches transit from Useless/Remote to Useless/Conflict/Remote. In addition, we see the two effects by considering the same reasons of the above discussion. First, the possibility for transiting from Useless/Conflict to Useless/Conflict/Remote is increased or decreased with the increase in the cache size. Second, the number of prefetches which transits to Useful/Conflict is increased, and the performance improvement can be expected for both

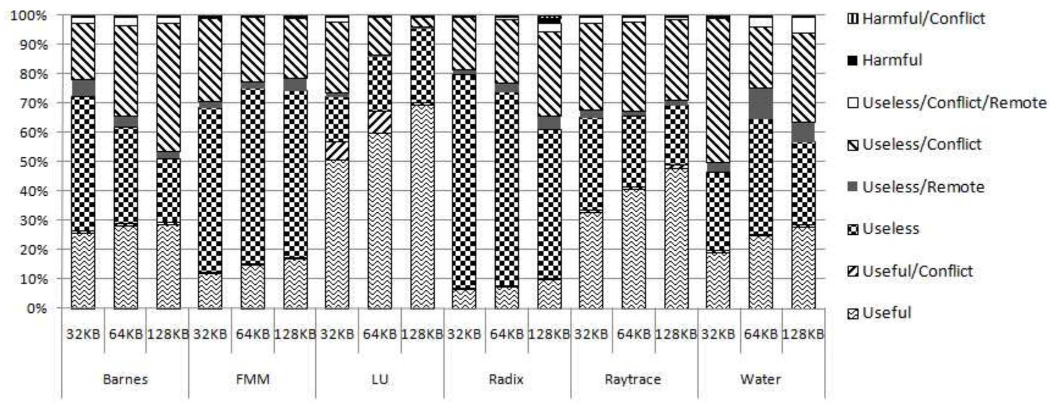


Figure 9. Breakdown of Prefetches with Tagged Prefetch. Varying L1 Cache Size from 32KB to 128KB.

local and remote cores.

We also discuss briefly the impact of the cache size for other prefetches. In all benchmarks, as the L1 cache size becomes larger, the percentage of Useful prefetches is increased, and the percentage of Useless prefetches is decreased due to longer lifetime of prefetched blocks. On the other hand, the rate of Useless/Conflict prefetches increases or decreases with respect to the benchmarks. In *LU*, Useless/Conflict state is dramatically reduced. This is caused by low possibility of evicting effective blocks with a large cache size. On the other hand, in *Barnes* and *Radix*, the rate of Useless/Conflict state is increased with an increasing the cache size. The reason for this decrease is high possibility of conflict events (*Event4*) due to the long time period of prefetched blocks in the L1 cache.

5 Conclusions

In this paper, we have proposed an extended taxonomy for classifying prefetches on CMPs, and have discussed quantitatively the impact of prefetching on memory performance. In our analysis, we have mainly found three key points. First, the conventional prefetch algorithms, which are developed for single-core microprocessors, do not exploit effectively the feature of CMPs. Second, the effects of prefetching differ vastly between multiprocessors systems and CMPs having a shared last-level cache. In the CMPs, the negative effect of invalidation for prefetched blocks is very small. Third, cache size to keep coherence between processor cores dramatically affect the impact of data prefetching on memory performance. Since a larger cache size expands lifetime of prefetched blocks in the cache, the possibility of references of prefetched blocks and invalidations of a prefetched blocks and increased.

For future work, we plan to analyze the effects of data prefetching on execution time and bus contentions in various simulation parameters. We also target to develop data prefetching algorithms to reduce memory access time not only in a local processor core but also in remote processor cores.

Acknowledgments

We thank all members of the System LSI laboratory of Kyushu University for discussing at technical meetings. This work has been supported by the Grant-in-Aid for Creative Scientific Research (KAKENHI) No.19200004, and Matsushita Electric Industrial Co. Ltd.

References

[1] J. L. Baer and T. F. Chen. An Effective On-Chip Preloading Scheme to Reduce Data Access Penalty. In Proceedings of the 1991 Conference on Supercomputing, June 1991.

[2] N. L. Binkert, E. G. Hallnor, and S. K. Reinhardt. Network-oriented Full-system Simulation Using M5. In Proceedings of the Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads, February 2003.

[3] N. D.Enright Jerger, E. L. Hill, and M. H. Lipasti. Friendly Fire: Understanding the Effects of Multiprocessor Prefetching. In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, March 2006.

[4] A. J. Smith. Cache Memories. Computing Surveys, Vol.14, No.3, September 1982.

[5] V. Srinivasan, E. S. Davidson, and G. S. Tyson. A Prefetch Taxonomy. IEEE Transactions on Computers, Vol.53, No.2, February 2004.

[6] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In Proceedings of the 22nd International Symposium on Computer Architecture, June 1995.