Design Considerations of High Performance Data Cache with Prefetching

Chi-Hung Chi, Jun-Li Yuan

School of Computing

National University of Singapore

Singapore 119260

Email: chich@comp.nus.edu.sg

Abstract. In this paper, we propose a set of four load-balancing techniques to address the memory latency problem of on-chip cache. The first two mechanisms, the **sequential unification** and the **aggressive lookahead** mechanisms, are mainly used to reduce the chance of partial hits and the abortion of accurate prefetch requests. The latter two mechanisms, the **default prefetching** and the **cache partitioning** mechanisms, are used to optimize the cache performance of the unpredictable references. The resulting cache, called the **LBD (Load-Balancing Data)** cache, is found to have superior performance over a wide range of applications. Simulation of the LBD cache with RPT prefetching (Reference Prediction Table - one of the most cited selective data prefetch schemes [2,3]) on SPEC95 showed that significant reduction in the data reference latency, ranging from about 20% to over 90% and with an average of 55.89%, can be obtained. This is compared against the performance of prefetch-on-miss and RPT, with an average latency reduction of only 17.37% and 26.05% respectively.

1. Introduction

To improve the accuracy and coverage of data prefetching, current research emphasizes on the exploration of hybrid data address and value prediction. Data accesses in a program are partitioned into exclusive distinct reference classes, each of which is handled exclusively by one predictor/prefetch unit. Good examples of these predictors include linear memory reference predictor and pointer predictors.

With an accurate predictor and its supporting hardware/software, very good cache performance is expected. However, experiment showed that it might not be as simple as it appears. Even though the overall cache performance can be improved, there are still many data references whose access patterns are predicted accurately but are missing from cache [5]. This percentage ranges from a few percents to over 95%, with an average of 54%. In other words, about half of the cache misses are actually due to data references that can be predicted accurately! The overall cache effectiveness is further bounded by the access behavior of unpredictable references in cache, which contributes to the other half of the cache misses.

In this paper, we propose a set of four load-balancing mechanisms to make up for the discrepancy between the ideal and the observable performance of accurate prefetching. They are (i) sequential unification of demand and prefetch requests, (ii)

aggressive lookahead prefetching, (iii) default prefetching, and (iv) cache partitioning. The first two mechanisms are mainly used to reduce the chance of partial hits and the abortion of accurate prefetch requests by demand fetch requests while the latter two mechanisms are to optimize the cache performance of the unpredictable references. To ensure the performance gain, these mechanisms will only be triggered selectively, based on the level of confidence on the urgency, accuracy, and nature of the prefetched data. The resulting cache, called the **LBD (Load-Balancing Data)** cache, is found to have superior performance over a wide range of applications. Simulation of the LBD cache with RPT prefetching (Reference Prediction Table [2,3]) on SPEC95 showed that significant reduction in the data reference latency, ranging from about 20% to over 90% and with an average of 55.89%, can be obtained. This is compared against the performance of prefetch-on-miss and RPT, with an average latency reduction of only 17.37% and 26.05% respectively. To help the discussion in the rest of the paper, the reference prediction table (RPT) prefetching [2-4] will be chosen as the basic accurate data prefetch scheme, on which our mechanisms will be added on. RPT is chosen because it is one of the most cited schemes for accurate data prefetching.

Before we proceed, it will be helpful to give precise definition for the following terms used in the rest of the paper. A *partial hit* for a cache block *I* is said to occur if, while the block *I* is being prefetched from the next level of the memory hierarchy, a demand fetch for block *I* occurs. A *selective prefetch (SP) scheme* refers to a scheme that is based on one or more reference predictors for data prefetching. Furthermore, it will only handle one type of data sequences with some predefined characteristics such as the linear memory accesses. In the rest of this paper, the term SP and RPT might sometimes be used interchangeably because RPT is chosen as the SP scheme here.

1.1. Basic Simulation Environment

<u>Processor Specification:</u> UltraSPARC ISA compatible Superscalar with 3 integer units, 3 floating point units, 1 branch unit, 1 LOAD/STORE unit Out-of-order execution, register renaming, 2-bit branch predictor, reorder-buffer with 64 entries.	
<u>1st Level Instruction Cache:</u> Cache Size: 32 Kbytes Block Size: 32 bytes Associativity: direct-mapped Replacement: LRU Placement: write-back Access Time to 2 nd Level Cache: 6 cycles Bandwidth to 2 nd Level Cache: 8 bytes Memory queue: 8 entries Basic prefetch-on-miss scheme	<u>1st Level Data Cache</u> Cache Size: 32 Kbytes Block Size: 32 bytes Associativity: direct-mapped Placement: write-back Mem. Access Time to 2 nd Level Cache: 6 cycles Bandwidth to 2 nd Level Cache: 8 bytes Memory queue: 8 queue Store buffer: 8 entries Non-blocking cache Prefetching using Chen's RPT (i.e. linear memory access prefetching) with 512 entries & lookahead of 3 iterations
<u>2nd Level Unified Cache:</u> Cache Size: 256 Kbytes Block Size: 64 bytes Associativity: direct-mapped	
Placement: write-back Mem. Access Time: 60 cycles Bandwidth to Main Memory: 8 bytes	

Table 1: Simulation Parameters for the Baseline Architecture

In our study, SPEC95 was chosen as the benchmark suite for experimentation. Each of these programs was traced and simulated cycle-by-cycle for 100 million instructions. The baseline architecture together with its parameters used in the simulation study is given in Table 1. It is an UltraSPARC ISA compatible superscalar processor with rich details on its architectural features, including the pipeline, register renaming, reorder buffer, branch prediction, and multi-level memory hierarchy.

2. Previous Research

Recent work in data prefetching concentrates on the predictability of references and the accuracy of prefetching. A direct consequence of this research direction is the introduction of hybrid address and value predictors [1,6,7,10,12]. Multiple predictors are implemented, each of which will be designed, optimized, and responsible for one type of references. Usually, they can achieve very high prefetch accuracy, but their coverages are quite restricted. Furthermore, there is always a non-negligible group of unpredictable references that they cannot handle. To overcome this problem, multiple prefetch schemes in a cache system are proposed [11]. In addition to the selective scheme(s), the prefetch-on-miss mechanism will be employed as the default prefetch scheme in case the references do not fall into the selective coverage set. Good results were reported. However, their interactions with other caching techniques such as cache partitioning/buffering still need to be studied.

Another technique in caching is the prefetch buffer. There are three interesting data buffer designs that are worth mentioning here. The first one is the HP's PA7200 multimedia processor [9]. Between the core processor and the off-chip cache, there is a data buffer, called the assist cache, on the processor chip. This assist cache is a 64 fully associative cache of size 2 Kbytes. All referenced data, both prefetched and demand-fetched, are moved into the assist cache. When data are replaced from the assist cache, there are supports (both in hardware and software) that allow the replaced data to bypass the off-chip cache. This is to reduce the interference between the prefetched spatial data and the temporal data in the off-chip cache.

The second one is the victim cache proposed by Jouppi [8]. A small buffer is inserted between the first and second level caches. When a cache block is replaced, it will be placed in the victim cache. This increases the "pseudo" associativity of the cache with much less hardware. Despite the performance improvement obtained, both designs do not address the problem of interference between spatial, prefetched data and temporal demand-fetched data.

Finally, in the same paper [8], a stream buffer is also described. The purpose of the stream buffer is to prefetch a number of linear memory reference sequences into a separate buffer instead of entering the cache. In this aspect, it functions like a prefetch buffer. However, the buffer design does not have any load-balancing mechanism to tradeoff space for bandwidth when partial hits or complete cache misses occur to the predictable reference group.

3. Load-Balanced Data Cache

In the beginning of this paper, we pointed out that there are four aspects of performance loss due to memory accesses. Either the correctly predicted data cannot be prefetched in time for consumption or the accurate prefetch requests are aborted while they are being served. This observation is generally true, independent of the prefetch accuracy. For the unpredictable group of references, since they are ignored by the SP schemes, no performance improvement is expected. Even worse, their working set might be destroyed by the predictable group of references and this results in more cache misses.

To make up for this performance gap, four mechanisms are proposed to load balance the bus bandwidth and the cache space. They are: (i) sequential unification of prefetch and demand fetch requests, (ii) the aggressive lookahead prefetching, (iii) default prefetching, and (iv) cache partitioning with load-balancing. The resulting cache that supports all these mechanisms, is called the **Load-Balancing Data (LDB)** cache.

3.1. Sequential Unification

In the current memory hierarchy system, the time required to transfer a block of data can roughly be divided into two components: (i) the memory startup latency between the sending out of a request and the receiving of the first byte of data, and (ii) the actual transfer time for the subsequent data. As the bus width between successive memory levels is getting larger, the overall fetch time for a block of data is mainly dominated by the memory startup latency. Thus, it will be significant to the system performance if the startup time of some memory requests can be minimized.

Given a memory access instruction (MAI), two possible types of data references can occur. They are the demand fetch and prefetch requests. Thus, one good way to eliminate the startup overhead of the memory requests is to combine these two types of requests for the same MAI aggressively. Whenever the memory blocks containing the demand fetch and prefetch data are consecutive to each other, they can be combined to form a single request with double block size. Since the combined request is still considered as a "demand" fetch, one side effect is the increase in the bus usage priority of the prefetch portion.

There is one side-effect of the sequential unification mechanism that one needs to be careful. It is about the bus usage priority. High priority is good to prefetch requests with high accuracy, but is bad to incorrect prefetch requests. The additional time required for the transfer of the prefetch data is likely to be seen by the processor. If the benefit for prefetching them is not high (again related to accuracy), it might actually degrade the performance instead. Consequently, this mechanism is triggered only when both requests are missed from cache. We choose this situation because the startup overhead of the unified request is the same as the unavoidable demand fetch, but that of the prefetch request is eliminated.

3.2. Aggressive Lookahead

Current data prefetch schemes try to prefetch the next N^{th} iteration data while the current iteration datum is being referenced, where N is usually small. Under this situation, partial hits are found to occur quite often, especially in the situation of small stride linear memory accesses. This is not difficult to understand. The next block is prefetched only when the current referenced block is about to be used up. Thus, a memory latency of 1 to (*Cache_Miss_Penalty* - 1) cycles might potentially be visible to the program execution for each partial hit. The abortion of accurate, on-going prefetch requests by demand fetch requests also contributes to some performance loss.

It is observed that during the time period of accessing two elements in the same cache block, the chance to have some free bus cycles is quite high. Thus, it might be useful to perform more aggressive, yet accurate prefetching during this period. This can be achieved easily through dynamic adjustment of the prefetch block address. When the current reference and the prefetch data are mapped to the same cache block address i , the prefetch block address will be adjusted to the next sequential one. This kind of additional prefetching has similar accuracy as the original schemes do because they go through similar kind of confirmation test. Note that unlike the sequential unification, its triggering will be independent of the reference hit or miss.

3.3. Default Prefetching

The coverage of a prefetch scheme refers to the portion of the data references in a program that the scheme tries to identify their reference patterns and to issue prefetch requests for them. Current SP schemes often trade off the prefetch coverage for accuracy and this puts an upper bound to their performance potentials.

To solve the coverage problem of the SP schemes, we propose a hybrid approach: highly accurate SP schemes should work co-operatively with less accurate (yet beneficial) prefetch schemes. Given a set of data references in a program, the cache prefetch unit divides them into two groups - a selected, predictable reference group and an unpredictable, non-spatial reference group. Data from the predictable reference group is handled by the SP scheme. The unpredictable reference group, instead of being ignored by the cache controller, is now handled by some default prefetch scheme such as POM. Additional performance improvement is possible because all data references in a program are now taken care of, although not to the same extent. Note that as far as each individual reference group is concerned, the prefetch accuracy obtained by its corresponding prefetch scheme is roughly unchanged.

3.4. Cache Partitioning and Prefetch Buffer

Interference among data references in cache refers to the situation when there is no free space for the incoming data and some data in cache needs to be replaced out.

In practice, it is impossible to eliminate all the data interference in cache because of the finite cache size. However, due to the difference in localities of the predictable and the unpredictable references, certain portion of the data interference in cache can be minimized through cache partitioning. Generally speaking, predictable data from the highly accurate SP schemes have stronger spatial locality while the unpredictable, demand fetched data have relatively stronger temporal locality. To minimize the interference among them, it is better to separate the predictable, spatial data prefetched by the SP schemes from the unpredictable temporal data that are fetched by demand or by the default prefetch schemes. This suggests the use of a separate prefetch buffer.

Further study on the design of the data prefetch buffer reveals that two more issues need to be taken care of. The first issue is the insufficient bandwidth for the SP prefetching. This results in partial hits and the memory latency cannot be completely hidden from the processor execution even if the prefetching were 100% accurate. The second issue is about the predictable data references that are accessed during the transient period when the hardware reference prediction table tries to build up its content; these data should not be prefetched into the prefetch buffer. To enhance the cache performance further, some kind of load balancing for the cache space usage is found to be useful: when a miss or a partial hit occurs to a predictable reference, the corresponding cache block will be re-directed from the prefetch buffer back to the data cache. This is to reduce the bandwidth demand when the bus is expected to be overloaded.

4. Experimental Result

To study the performance of our proposed mechanisms, cycle-by-cycle simulation on the baseline architecture was conducted for various cache configurations and control designs. In the analysis, the normal data cache without prefetching was used as the reference standard. And the memory latency reduction is defined by the following formula:

$$Mem_latency_reduction = \frac{Execution_Time(Normal_Cache_No_Prefetch) - Execution_Time(New_Design)}{Execution_Time(Normal_Cache_No_Prefetch) - Execution_Time(Perfect_Cache)} * 100\%$$

The overall result is summarized in Table 2. It shows that when the four load-balancing mechanisms are used collaboratively in the LBD cache, significant improvement in cache performance can be obtained for almost all benchmark programs simulated. Its reduction percentage ranges from about 20% to over 90%, with an average of 55.89%. This is compared to the performance of the POM and the RPT, with an average latency reduction of only 17.37% and 26.05% respectively. The huge performance improvement is actually within our expectation. In the POM case, its prefetch accuracy for the unpredictable reference group is the same as that of the LBD cache, but its accuracy for the predictable reference group is much lower than that of the LBD cache. In the RPT case, both the RPT and the LBD cache have similar performance for the predictable references, but the LBD cache takes care of the unpredictable reference group instead of ignoring them. In other words, the LBD

cache can handle both predictable and unpredictable reference groups well. Furthermore, its cache space and bandwidth are well balanced and the interference between the predictable and unpredictable reference groups is greatly reduced.

Overall Performance	POM	RPT					LB-Data Cache
		Original	+ Unification	+ Aggressive	+ POM	+ Pref. Buffer	
Latency Improv.	17.37%	26.05%	30.18%	25.69%	29.59%	44.47%	55.89%
Prefetch Req. Sent (w.r.t. POM)	100.00%	67.03%	66.66%	68.06%	124.47%	106.15%	164.62%
Prefetch Accuracy	37.62%	67.18%	67.32%	65.99%	53.41%	75.03%	77.22%

Table 2: Overall Performance of Various Prefetch Schemes on SPEC95

One very interesting observation is that while the combined effect of the four mechanisms in the LBD cache is very impressive, their individual effects differ significantly. Among the four proposed mechanisms, the load-balancing prefetch buffer is found to have superior performance over the other three. An extra performance of 18.42% memory latency reduction is obtained by the prefetch buffer, as compared to the 5% additional performance gain obtained by the others. This suggests that *while improving prefetch accuracy and performing more aggressive prefetching are important, the interference among the predictable and unpredictable references cannot be ignored*. For the other three mechanisms, their individual effectiveness seems to be quite limited. The default prefetching and the sequential unification mechanisms can only provide an additional performance of 3.54% and 4.13% respectively while the aggressive prefetching mechanism even performs slightly worse than the original RPT scheme does. This is surprising because this means that the overall performance gain of the LBD cache is not accumulated from the individual results. Further investigation found that they actually enforce the effect of each other instead. A separate study on the sequential unification and aggressive lookahead mechanisms only [5] also observed similar enforcement effect as we have here.

5. Conclusion

In this paper, we propose four mechanisms to improve the data cache performance. By integrating a prefetch request into an unavoidable demand fetch request, its startup memory latency can be eliminated and the bus priority of the prefetch request is increased. More aggressive prefetching is also carried out whenever the bus is expected to have relatively light workload. With default prefetching on top of the accurate SP prefetching, the unpredictable reference group can now be taken care of. Finally, the interference between the predictable and unpredictable reference groups in cache can be minimized with the help of a load-balanced prefetch buffer. Simulation shows that these four mechanisms enforce each other and provide significant performance improvement over current accurate prefetch schemes such as the RPT.

References

1. Black, B., Mueller, B., Postal, S., Rakvic, R., "Load Execution Latency Reduction," *Proceedings of the ACM International Conference on Supercomputing*, July 1998.
2. Chen, T.F., Baer, J.L., "Reducing Memory Latency via Non-Blocking and Prefetching Caches," *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1992, pp. 51-61.
3. Chen, T.F., Baer, J.L., "Effective Hardware-Based Prefetching for High Performance Processors," *IEEE Transactions on Computers*, Vol. 44, No. 5, May 1995, pp. 609-623.
4. Chen, T.F., "Data Prefetching for High Performance Processors," *Ph.D Thesis, Department of Computer Science and Engineering, Washington University*, 1993.
5. Chi, C.H., Yuan, J.L., "Sequential Unification and Aggressive Lookahead Mechanisms for Data Memory Accesses," *Internal Report, School of Computing, National University of Singapore*, 1999.
6. Gonzalez, J., Gonzalez, A., "Speculative Execution via Address Prediction and Data Prefetching," *Proceedings of the ACM International Conference on Supercomputing*, July 1997, pp. 196-203.
7. Ibanez, P., Vinals, V., Briz, J.L., Garzaran, M.J., "Characterization and Improvement of LOAD/STORE Cache-Based Prefetching," *Proceedings of the ACM International Conference on Supercomputing*, July 1998.
8. Jouppi, N.P., "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proceedings of the 18th Annual Symposium on Computer Architecture*, May 1990, pp. 364-373.
9. Kurpanek, G., Chen, K., Zheng, J., DeLano, E., Bryg, W., "PA7200: A PA-RISC Processor with Integrated High Performance MP Bus Interface," *IEEE Publications 1063-6390/94*, 1994, pp. 375-382.
10. Lipasti, M.H., Wilkerson, C.B., Shen, J.P., "Value Locality and Load Value Prediction," *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1996.
11. Manku, G.S., Prasad, M.R., Patterson, D.A., "A New Voting Based Hardware Data Prefetch Scheme," *Proceedings of 4th International Conference on High Performance Computing*, Dec. 1997, pp. 100-105.
12. Sazeides, Y., Smith, J.E., "The Predictability of Data Values," *Proceedings of the MICRO-30*, 1997, pp. 248-258.