International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 3 Issue 6, October 2019 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

A Taxonomy of Data Prefetching Mechanisms

Yee Yee Soe

Faculty of Computer System and Technologies, University of Computer Studies, Hpa-An, Myanmar

ABSTRACT
Data prefetching has been considered an effective way to cross the performance gap between processor and memory and to mask data access latency caused by cache misses. Data prefetching prefers data closer to a processor before it is actually needed with hardware and/or software support. Many prefetching techniques have been proposed in the few years to reduce data access latency by taking advantage of multi-core architectures. In this paper, a taxonomy that classifies various design concerns has been proposed in developing a prefetching strategy. The various prefetching strategies and issues that have to be considered in designing a prefetching strategy for multi-core processors.

KEYWORDS: *Data prefetching, cache misses, performance, taxonomy, multi-core processors*

How to cite this paper: Yee Yee Soe "A Taxonomy of Data Prefetching Mechanisms" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-3 | Issue-6, October 2019, pp.1122-1127, URL: https://www.ijtsrd.com/papers/ijtsrd29339.pdf

Copyright © 2019 by author(s) and International Journal of Trend in Scientific Research and Development Journal. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (CC BY 4.0) (http://creativecommons.org/licenses/by/4.0)

1. INTRODUCTION

A data prefetching strategy has to contemplate various issues in order to hide data access latency efficiently. It should be able to forecast future accesses precisely and to move the predicted data from its source to destination in time. Some proposed strategies predict following data accesses using recent history of data accesses from which patterns can be recognized [1][2][3], using compiler and user provided hints [4], analyzing traces of past execution of applications or loops [5], and running a helper thread ahead of actual execution of an application to predict cache misses [6][7]. Compiler and user provided hints are used in software level prefetching [8] [9].

Among these strategies, predicting future data accesses based on recent history has been popular and implemented at hardware level in existing processors. Helper-thread initiated prefetching is becoming popular in multi-threaded and multi-core processors. Prefetching data exactly in time is a challenging job. By considering what to prefetch and when to prefetch aspects, complexity of executing prefetching methods should be low in actual processing. Data should not be prefetched too late or too early. Data prefetching is a data access latency concealing technique, which decouples and overlaps data transfers and computation. The data prefetching predicts future data accesses, initiates a data fetch, and brings the data closer to the computing processor before it is requested to be reduce CPU stalling on a cache miss.

In Scenario A, a prefetch engine notices history of L1 cache misses and initiates prefetch operations. In multi-threaded and multi-core processors, pre-execution based approaches use a separate thread to predict future accesses (scenario B). A prefetching-thread pre-executes data references of a main computation-thread and begins prefetching data into a shared cache memory (L2 cache) earlier than the computation-thread. In memory-side prefetching strategy, (scenario C) the prefetching-thread is executed on an intelligent main memory, where a memory processor pre-executes helper-threads. The predicted data is pushed towards the processor. From these scenarios, it is evident that in addition to predicting 'what' and 'when' to prefetch, sources, destinations, and initiators of prefetching play primary role in designing an effective prefetching strategy.

In this paper, a taxonomy of prefetching strategies that primarily captures design issues of prefetching strategies. The definitions of prefetching and compared various prefetching strategies in the context of single-core processors have been discussed in their surveys[10][11]. Their consideration provides a taxonomy addressing 'what', 'when', and 'where' (destination of prefetching) questions for hardware prefetching and software prefetching. The appearance of multi-thread and multi-core processor architectures brought new opportunities and challenges in designing effective prefetching strategies. A taxonomy of prefetching mechanisms based on a comprehensive study of hardware prefetching, software prefetching, prediction and pre-execution based prefetching, and more importantly strategies that are novel to multi-core processors has been proposed.

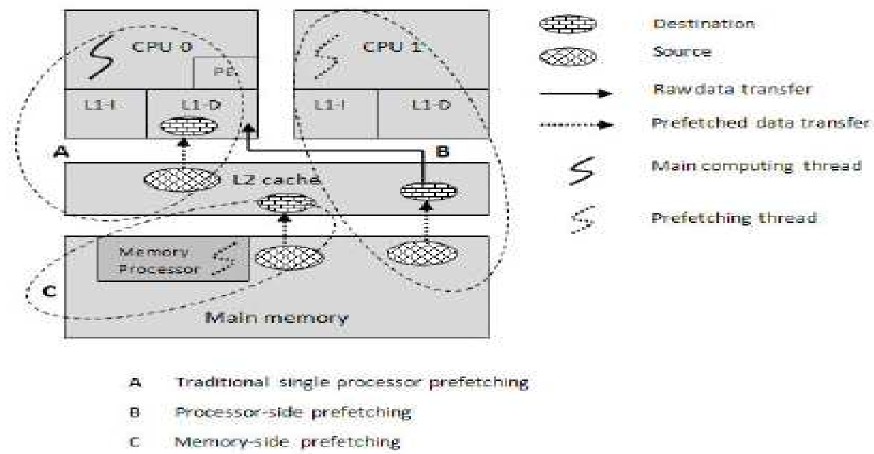

Figure-1 Prefetching Scenarios

2. Taxonomy

The top-down approach to characterize and classify the design issues of prefetching strategies has been taken. Figure-2 shows the top layer of the classification, which consists of the five most fundamental issues that any prefetching strategy has to address: what data to prefetch, when to prefetch, what is the prefetching source, what is the prefetching destination, and who initiates a prefetch.

Figure2. Five fundamental issues of prefetching

2.1. What to prefetch?

Predicting what data to prefetch is the most significant requirement of prefetching. If a prefetching strategy can predict the event of future misses, then prefetch instructions can be issued ahead and bring that data by the time the cache misses. To hide the stall time caused by cache misses effectively, the accuracy of predicting what to prefetch must be high.

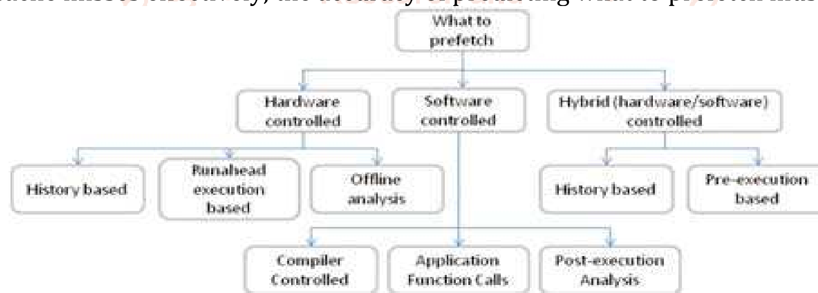

Figure-3 Prediction What Data to Prefetch

Predicting future data references accurately is critical. Low precision leads to cache pollution. A classification of predicting what data to prefetch based on where it is implemented and on various techniques as shown in Figure-3.

2.1.1. Hardware controlled strategies.

In hardware controlled data prefetching, prefetching is executed in hardware. Many methods support hardware controlled prefetching.

History-based prediction is the most often used among hardware controlled data prefetching strategies. In these strategies, a prefetch engine is used to predict future data references and to provide prefetching instructions. All the components of prefetching are executed within a processor and they do not require any user interference. A prefetch engine notices the history of data accesses or the past of cache misses to predict later accesses by a processor. The different algorithms used in history based prediction have discussed.

Run-ahead execution uses idle cycles or cores to run instructions while a CPU is stalled or idle. The dual-core execution approach uses an idle core of a dual-core processor has suggested to construct large, distributed instruction window [12] and future execution uses an idle core to pre-execute following loop iterations using value prediction [13].

Off-line analysis strategy is another hardware controlled prefetching approach. A method has proposed [5], where data access patterns are analyzed for hotspots of code that are frequently executed. This approach works well for requests that refer to similar data access pattern.

2.1.2. Software controlled strategies.

Software-controlled prefetching [8] [9] allows control to a developer or to a compiler to put prefetching instructions into programs. Software-controlled prefetching can use compiler controlled prefetching instructions or function calls in source code or prefetching instructions inserted based on post execution analysis have shown in figure-3. Many processors provide support for prefetching instructions in their instruction set. Compilers or application developer can place these prefetch instructions or built-in procedures provided by compilers. Software-controlled prefetching puts burden on developers and compilers, and is less effective in overlapping memory access stall time on ILP processors due to late prefetches and resource argument [14]. Post-execution analysis can be used, where marks of data accesses are analyzed for patterns.

2.1.3. Hybrid hardware/software controlled strategies.

Hybrid hardware/software controlled strategies are obtaining popularity on processors with multi-thread support. On these processors, threads can be used to move complex algorithms to predict following access patterns. These methods require help from hardware to run helper-threads that are specifically executed to prefetch data. They require software aid to synchronize with the actual computation thread. The helper-thread based prefetching strategies either examines the past of data accesses of a computation thread or pre-execute data thorough parts of the computation thread that warms up a shared cache memory by the time a raw cache miss happens.

History based hybrid prediction strategies [14] inspect the past of accesses to predict following accesses and prefetch data. Pre-execution based techniques [15] using a helper-thread to accomplish slices of code ahead of computation thread.

2.1.4. History-based prediction algorithms.

Hardware-controlled, software-controlled, and hardware/software controlled proposals use prediction algorithms based on the past of data accesses or cache misses have shown in figure-3. Prediction algorithms look for different patterns one of the past of data entries. Figure-4 shows a classification of data access designs based on spatial space between entries, their repeating behavior and request size of accesses. Spatial patterns are divided based on the number of bytes (also called as stride) between consecutive accesses as contiguous, non-contiguous, and combinations of both. Non-contiguous designs are classified by the quality of strides between accesses. Data access patterns repeat when loops or functions execute frequently. These patterns as either one event or repeating patterns have arranged. Request dimension in each access may be fixed or variable. This classification communicates a comprehensive scope of data accesses.

Some prediction algorithms have suggested occurring these designs. Sequential prefetching [2] gets neighboring cache blocks by taking advantage of location. Stride prefetching proposal [1] predicts the following accesses based on strides of the new archive. Stride prefetching strategies support a reference prediction table (RPT) to keep path of new statistics accesses. To express repetitiveness of data accesses, Markov prefetching [3] was suggested. Distance prefetching [16] uses Markov chains to build and keep probability transition diagram of strides among data accesses. Multi-level Difference Table (MLDT) [17] uses time-series analysis method to forecast the past accesses in a sequence, by finding the differences in a sequence to multiple levels.

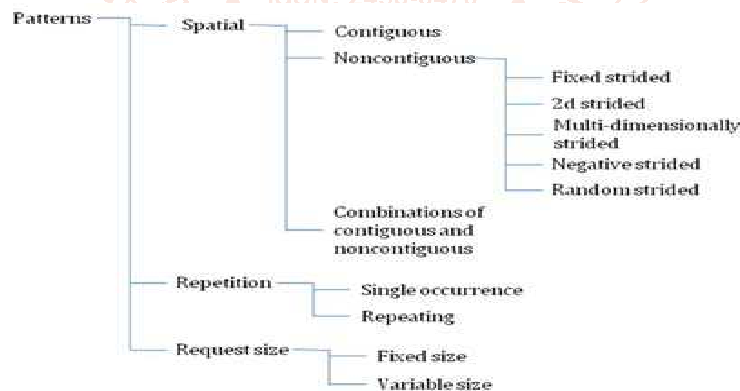

Figure4. Classification of data access design

2.2. When to prefetch?

The time to issue a prefetch order has notable result on the common execution of prefetching. Prefetched data should happen its destination before a fresh cache miss takes place. The sequence of timely prefetching depends on complete prefetching above (i.e. the above of predicting later accesses plus the above in prefetching data) and the time for the occurrence of following cache failure. If the complete prefetching above exceeds the time of succeeding cache miss, modifying prefetching space can keep away from late prefetches. Figure-5 shows a classification of different methods used in determining when to prefetch. Occurrence based process issues a prefetch instruction on some happening, such as a memory reference or a cache miss or a branch or accessing a previously prefetched data chunk for the first time. Prefetching on each memory reference is also called every time prefetch. Prefetch on failure is an application on existing processors as it is easy to apply.

Tagged prefetching [3] begins a prefetch command when a data entry hits previously prefetched data block for the first time. Branch directed prefetching [3] proposes that since branch instructions decide which instruction path is followed, data access samples are also dependent upon branch instructions. Chen et al. [4] have proposed using a lookahead program counter (LA-PC).

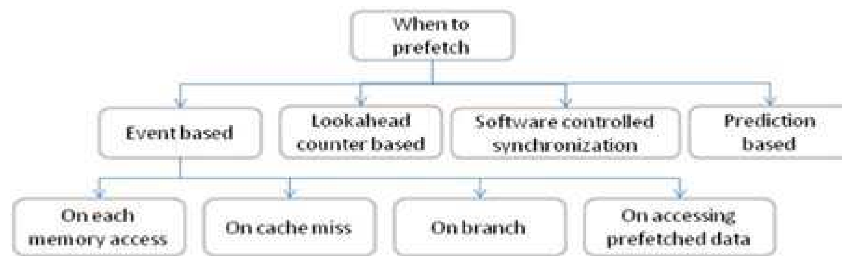

Figure5. Predicting 'when' to prefetch

In loop codes, as an alternative prefetching one iteration forwards, the lookahead prediction alters prefetching interval using a pseudo counter, called LA-PC that remains a few cycles leading of real PC.

Software-controlled prefetching approaches need either compiler or application developers to build decision to put prefetching tasks sufficient to prefetch data. An algorithm to compute prefetching interval [8] has provided by Mowry et al. [9]. According to this algorithm, prefetching commands are called severely for data references that would origin cache misses. In helper-thread based approaches, regular synchronization of computation drift with helper-thread is needed to stop late prefetches or very early prefetches. The synchronization to stop helper-thread execution lagging behind computation thread has used by Song et. al.[6]. In many applications, data access bursts follow sure design. By examining the time intervals, following data bursts can be predicted to begin prefetching. Server-based push prefetching [17] uses prediction based strategy to inspect when to prefetch.

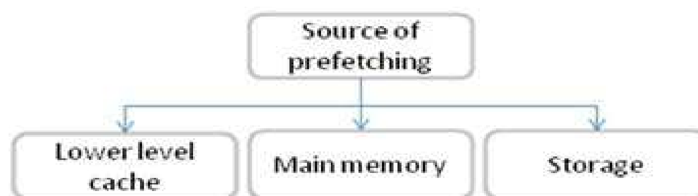

Figure6. Source of prefetching

2.3. What is the source of prefetching?

Memory ranking holds multiple levels consisting cache memories, main memory, secondary storage, and tertiary storage. Data prefetching can be applied at different levels of memory ranking. Statistics can be prefetched between cache memories and main memory, or between main memory and storage as shown in Figure-6. To design a prefetching strategy, it is requisite to realize where the newest duplicate of data is. In existing deep memory hierarchies with write back strategy, data can live at any level of memory ranking. In single-core processors, prefetching origin is generally the main memory or lower level cache memory. In multi-core processors, memory hierarchy carries local cache memories that are private to each key and cache memories that are shared by multiple cores. Planning a prefetching strategy considering several copies of a data in local cache memories may guide to data consistency concerns, which is a challenging job. The focus of this conversation is restricted to cache and memory level prefetching.

2.4. What is the destination of prefetching?

Destination of prefetched data is another vital concern of prefetching strategy. Prefetching destination should be closer to CPU than a prefetching source in order to get performance benefits. Data can be prefetched either into a cache memory that is local to a processor or into a cache memory that is shared by multiple processing keys, or to a separate prefetch cache has shown in Figure-7. A separate prefetch cache can be either private to a processor core or shared by multiple keys.

While the greatest destination of prefetching facts is the private cache to keep away from cache pollution, there are many pattern issues that influence such prefetch master plan. One of them is the small dimensions of cache memory. Prefetching facts into this cache may cause cache pollution. To reduce the cache pollution, a dedicated buffer called prefetch cache was suggested. In multi-core processors, destination of prefetching changes. Each key may prefetch facts to its private cache or its private prefetch cache. One more outline is that one of the keys prefetches statistics into a shared cache (e.g. helper-thread based pre-execution). A Prefetch Buffer Filter (PBF) has suggested by Casmira et al. [18].

Figure7. Destination of prefetching

2.5. Who initiates prefetch instructions?

Prefetching commands can be issued either by a processor which needs data or by a processor that gives such a service. The first procedure is also called 'client-initiated' or 'pull-based' prefetching and the latter is called 'push-based' prefetching. Figure-8 shows a further classification of 'pull-based' and 'push based' strategies depending on where the initiator is discovered.

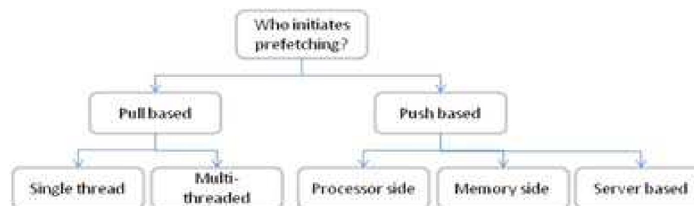

Figure8. Initiator of prefetching

Pull-based prefetching has been a usual approach of prefetching in single-core processors. In this technique, prefetching procedure (prediction and initiation) is within the processor. Multi-threaded processors enable decoupling of data entry from calculating. Helper-thread based prefetching [15] is a pair of typical helper-thread based prefetching strategies that pull data closer to a processor from main memory.

In Push-based prefetching, a key other than the real computation core fetches data. Run-ahead execution [12] and helper-thread based prefetching methods [15] also can be run on an unrelated key on processor side to push data into a shared cache, which is used by the computation core.

Memory-side prefetching is comparatively a new idea, where a processor occupying in the main memory pushes predicted data near to the processor [17]. Server-based push strategy pushes statistics from its source to destination without waiting for requests from processor side. Data Push Server (DPS) [20] uses a dedicated server to begin and proactively pushes data closer to the client in time.

Both (pull and push) techniques have pros and cons. Pull-based prefetching is restricted by complexity. In pre-execution based prefetching with the use of helper-threads, synchronization is required to begin pre-execution. Intuitively, with the assumption of same prediction overhead and same accuracy as those of client-initiated prefetching, push based prefetching is better than pull-based prefetching procedure since push based prefetching moves the complexity of prefetching outside the processor. Another profit is that push based prefetching is faster as main memory does not have to wait for a prefetching request from the processor. However, scalability of a memory processor becomes an issue when many processing cores have to be served in memory side prefetching. Server-based push prefetching solves this difficulty by using dedicated server cores.

3. Prefetching for Multicore Processors

Scheming prefetching strategies for multi-core processors poses new challenges. These challenges include multiple computing cores' competing to fetch regular data and prefetched data, while sharing memory bandwidth. With single-core processors, main memory receives prefetching requests just from one core. In multi-core processors, prefetching requests from multiple cores may set more force on main memory in addition to uniform data get requests. For example, the memory processor based solutions [14] are not scalable to monitor data access history or pre-execute threads and predict future references for multiple cores. This problem can be solved by decoupling data access from calculating cores. In Server-based push prefetching [19], a dedicated server core to provide data access support has proposed by predicting and prefetching data for computing cores.

Another challenge of multi-core processor prefetching is cache consistency. Multi-core processors retrieve the main memory, which is shared by multiple cores and at some level in the memory hierarchy they have to decide conflicting accesses to memory. Cache consistency in multi-core processors is distributed either by directory-based approach or by snooping cache accesses. Prefetching requests to shared data can be dropped to reduce complexity of coherence.

Usage of aggressive prediction algorithms on single-core processors has been discouraged as their complexity may become counter productive. With large amount of computing available, transferring complexity to idle or dedicated cores using Server-based push prefetching architecture [19] is beneficial.

4. Conclusions

Performance obtains of prefetching strategies depend on different criteria. With the emergence of multi-core and multi-threaded processors, new challenges and issues need to be considered to prefetch data. In this paper, a taxonomy of the five primary issues (what, when, destination, source, and initiator), that are necessary in designing prefetching strategies has provided. Each issue in detail, which defines the design of a prefetching strategy using examples of various prefetching strategies has discussed. The challenges of prefetching strategies in multi-core processors have also discussed. To be effective, a prefetching strategy for multi-core processing environments has to be adaptive to select among multiple procedures to forecast future data accesses. When a data access pattern is easy to be establish, prefetching strategy can choose history-based prediction algorithms to predict future data accesses. If data accesses are random, using pre-execution based approach would be advantageous. The server-based push prefetching selects prediction strategies considering these challenges.

5. References

[1] T. F. Chen and J. L. Baer, "Effective Hardware-Based Data Prefetching for High *Performance Processors" IEEE Transactions on Computers, pp. 609-623, 1995.*

[2] F. Dahlgren, M. Dubois, and P. Stenström, "Fixed and Adaptive Sequential Prefetching in Shared-memory Multiprocessors," *Proceedings of International Conference on Parallel Processing*, pp. 156-163, 1993.

[3] D. Joseph and D. Grunwald, "Prefetching Using Markov Predictors", *Proceedings of the 24th International Symposium on Computer Architecture*, pp. 252-263, 1997.

[4] C. K. Luk and T. C. Mowry, "Compiler-based Prefetching for Recursive Data Structures", *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996.

[5] J. Kim, K. V. Palem and W.F. Wong, "A Framework for Data Prefetching using Off-line Training of Markovian Predictors", *Proceedings of the 20th International Conference on Computer Design,* 2002.

[6] Y. Song, S Kalogeropulos and P Tirumalai, "Design and Implementation of A Compiler Framework for Helper Threading on Multi-Core Processors", *Proceedings of the 14th Parallel Architectures and Compilation Techniques*, pp. 99-109, 2005.

[7] C. Zilles and G. Sohi, "Execution-based Prediction Using Speculative Slices", *in Proceedings of the 28th International Symposium on Computer Architecture*, 2001.

[8] A. C. Klaiber and H. M. Levy, "An Architecture for Software-controlled Data Prefetching", *Proceedings of the 18th International Symposium on Computer Architecture*, pp.43-53, 1991.

[9] T. Mowry and A. Gupta, "Tolerating Latency through Software-controlled Prefetching in Shared-memory Multiprocessors", *Journal of Parallel and Distributed Computing*, 12(2), pp.87-106, 1991.

[10] N. Oren, "A Survey of Prefetching Techniques", TR CS-2000-10, University of the Witwatersrand, 2000.

[11] S. VanderWiel and D.J. Lilja, "Data Prefetch Mechanisms", *ACM Computing Surveys*, 32(2), 2000.

[12] H. Zhou, "Dual-Core Execution: Building a Highly Scalable Single-Thread Instruction Window", *Proceedings of the 14th Parallel Architectures and Compilation Techniques*, 2005.

[13] I. Ganusov and M. Burtscher, "Future Execution: A Hardware Prefetching Technique for Chip Multiprocessors", *Proceedings of the 14th Parallel Architectures and Compilation Techniques*, 2005.

[14] Y. Solihin, J. Lee and J. Torrellas, "Using a User-Level Memory Thread for Correlation Prefetching", *Proceedings of the 29th International Symposium on Computer Architecture*, pp. 171-182, 2002.

[15] C. K. Luk, "Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors", *Proceedings of the 28th International Symposium on Computer Architecture*, 2001.

[16] G. Kandiraju and A. Sivasubramaniam, "Going the Distance for TLB Prefetching: An Application-Driven Study", *Proceedings of the 29th International Symposium on Computer Architecture*, 2002.

[17] Xian-He Sun, Surendra Byna, and Yong Chen, "Server-based Data Push Architecture for Multi-processor Environments", Journal of Computer Science and Technology (JCST), 22(5), pp. 641-652, 2007.

[18] J. P. Casmira and D. R Kaeli, "Modeling Cache Pollution", *International Journal of Modeling and Simulation*, 19(2), pp. 132-138, 1998.

[19] X. H. Sun, S. Byna and Y. Chen, "Improving Data Access Performance with Server Push Architecture", *Proceedings of the NSF Next Generation Software Program Workshop (with IPDPS '07)*, 2007.

[20] C. L. Yang, A. R. Lebeck, H.W. Tseng and C. Lee, "Tolerating Memory Latency through Push Prefetching for Pointer-Intensive Applications", *ACM Transactions on Architecture and Code Optimization*, 1(4), pp. 445-475, 2004.