ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

An Energy Efficient Analysis of Hardware Prefetching Techniques:A Review

Mouneshwar Kanamadi

Department of CSE, Ashokrao Mane Group of Institutions, Vathar tarf vadgaon,

Tal. Hatkanangle, Dist. Kolhapur, India

ABSTRACT: The paper presents a review on different hardware prefetching techniques. It is written to be accessible to researchers familiar to hardware prefetching. Both the historical basis of the field and a broad selection of current work are summarized. Memory latency and bandwidth are progressing at a much slower pace than processor performance. A widely explored approach to improve cache performance is hardware prefetching that allows the pre-loading of data in the cache before they are referenced. Data prefetching has been considered as an effective way to mask data access latency caused by cache misses and to bridge the performance gap between processor and memory. Different hardware architecture and prefetching patterns are considered in this paper. Some of the energy preservation schemes are discussed and the results obtained from different methods are given in brief.

Keywords: Prefetching, Context Based Prefetching.

I. INTRODUCTION

Data prefetching is a data access latency hiding technique. Data prefetching has been considered as an effective way to mask data access latency caused by cache misses and to bridge the performance gap between processor and memory. With hardware, data prefetching brings data closer to a processor before it is actually needed. In order to reduce CPU stalling on a cache miss, data prefetching predicts future data accesses. Many prefetching techniques have been proposed in the last few years to reduce data access latency by taking advantage of newer architectures. A data prefetching strategy has to consider various issues in order to mask data access latency efficiently. It should be able to predict future accesses accurately and to move the predicted data from its source to destination in time. This hardware prefetching has to be carried out in the energy efficient manner, so that all these strategies can be applicable in the mobile device and the embedded system. In hardware prefetching, the hardware alone decides what data to prefetch and when and where to prefetch the data.

In hardware controlled data prefetching, prefetching is implemented in hardware. Various methods support hardware controlled prefetching.

The time to issue a prefetch instruction has significant effect on the overall performance of prefetching. Prefetched data should arrive to its destination before a raw cache miss occurs.

The efficiency of timely prefetching depends on total prefetching overhead (i.e. the overhead of predicting future accesses plus the overhead in prefetching data) and the time for the occurrence of next cache miss.

Destination of prefetched data is another major concern of prefetching strategy. Prefetching destination should be closer to CPU than a prefetching source in order to obtain performance benefits. Prefetching instructions can be issued either by a processor which requires data or by a processor that provides such a service. History-based prefetching is one of the basic methods- according to it data access history or cache miss history is used to predict future data access. This method has been proved effective for regular data access patterns, but for irregular codes, it's difficult, inaccurate, it not only can not improve the performance but also adds the overhead of cache miss. So different newer methods are proposed to avoid these overheads.

The remainder of this paper will be organized as follows. First we present the hardware prefetching by context based prefetching [1], then by dynamic multi-core hardware prefetching technology [2], then we present some energy schemes of hardware prefetching by using heterogeneous interconnects [3] and some complier and new power aware prefetching engine [4] in the section 2. In section 3, we deduce a conclusion.

II. RELATED WORK

First there are two hardware prefetching techology and then we have two energy efficient strategies: Improving the Effectiveness of Context-based Prefetching

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

with Multi-order Analysis[1], Analysis and improvement of dynamic multi-core hardware prefetch technology based on pre-execution[2], Energy-Efficient Hardware Prefetching for CMPs using Heterogeneous Interconnects[3] and Energy-Efficient Hardware Data Prefetching[4].

A. *Improving the Effectiveness of Context-based Prefetching with Multi-order Analysis.*

There exist many commercial high-performance data prefetching techniques, among them, the context based data prefetching has received attention in recent years due to its general applicability and high accuracy. There exist the limitation on the context-based prefetching that most of them are single order context analysis and prediction. Motivated by the observations, the new context-based prefetching method named Multi-Order Context-based (MOC) prefetching to address the drawback of existing context-based prefetching and to increase the overall prefetching effectiveness.

The essential idea of the context-based data prefetching is to detect the correlation between current context (the miss access information) and the past history and make predictions for data prefetching. A context-based prefetching method usually builds a state transition diagram with the access address strides (deltas) as states, and characterizes the correlation among miss address streams. Depending on the length of the context considered, the prefetching strategy can adjust the prefetching overhead and confidence in prefetching.

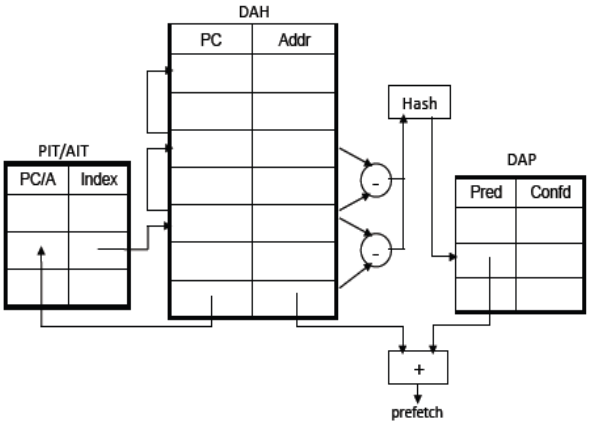

Fig. 1. Multi-Order Context-based Prefetching

StrategyThe mechanism of multiple order context-based prefetching is described as follows. This multi-order context-based prefetching has a three-level table organization as shown in Fig. The first two levels follow the design of the Data-Access History Cache (DAHC), which includes a Data Access History (DAH) table and index tables, PC Index Table (PIT) and Address Index Table (AIT). The DAH table stores the detailed data access history information. It gives high priority to recent access history, and thus filters outdated history automatically.

The index tables provide the view and the detection of the correlations from the instruction stream and address stream respectively. The third level table is the Data Access Prediction (DAP) table. Each DAP entry includes two fields. The first one stores the predicted address corresponding to the context indicated by the entry index. The second field stores a confidence counter that is used to represent how strong the prediction is. DAP entries are indexed by the hashed form of the contexts derived from a hash function. In the current design, they select FS-5 hash function to produce the hashed context as it has better performance than other choices. The order-1 prefetching is similar to the distance prefetching based on DAHC. The order-2 prefetching uses the sequence of strides between miss addresses as the context as shown in Fig. When a new data access is captured in DAH, the prefetcher searches the PIT to find the last miss address from the same PC. It then follows the PC chain to retrieve the miss sequence from the same PC.

On average, MOC reduces L2 cache misses by over 65%, which is more than two times of what the distance prefetching can achieve. Cache misses are reduced significantly for most benchmarks including five benchmarks whose misses are reduced over 90%.

B. *Analysis and improvement of dynamic multi-core hardware prefetch technology based on pre-execution.*

Multi-core processor has been widely used for lower power consumption and high performance, but it also aggravates the “Memory Wall” problem, the increasing memory access latency limits the further improvement of multi-core’s performance. So here three important strategies of prefetching are given such as software-controlled strategies, hardware-controlled strategies, hybrid hardware/software-controlled strategies, where Softwore-controlled prefetching Software-controlled strategies enables a programmer or a compiler to insert prefetching instructions into programs by exploiting the applications memory access patterns, hardware prefetching consists of two methods such as History-based prefetching*:* According to which data access history or cache miss history are used to predict future data access. This method has been proved effective for regular data access patterns, but for irregular codes, it’s difficult, inaccurate, it not only can not improve the performance but also adds the overhead of cache miss, and another method is Pre-execution-based prefetching, this mechanism uses a dedicated thread or an idle core of a

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

CMP to prefetch data for the thread or cores running the main program, this method are beneficial to applications with regular or random accesses. And the last method is Hybrid hardware/software-controlled strategies Equations, these strategies are gaining popularity on processors with multi-thread support. On these processors, threads can be used to run complex algorithms to predict future accesses. These methods require hardware support to run threads that are specially executed to prefetch data.

The main components of hardware prefetching involves

a) Run-ahead execution: This method allows microprocessors pre-process instructions during cache miss cycles instead of stalling.

b) FE(Future Execution): It use an idle core to prefetch data for another core running program.

c) DCE(Dual-core execution): This technique consists of two superscalar cores (a front and back core) coupled with a queue, the front processor fetches and preprocesses instruction streams and retires processed instructions into the queue for the back processor to consume.

Run-ahead execution is the first proposed prefetching technique based on out-of-order execution, DCE(Dual-core Execution) is motivated partly from run-ahead execution, and if both run-ahead execution and FE(Future Execution)are employed together, their cumulative effect is quite impressive.

Run ahead Execution: Principle- The concept of run-ahead execution was first proposed for in-order processors and further extended to perform prefetching for out-of-order architectures. In run ahead execution, when an instruction window is blocked by a long latency cache miss, the state of the processor is check pointed and the processor enters the 'run-ahead' mode by providing an invalid result for the blocking instruction and letting it graduate from the instruction window. In this way, the processor can continue to fetch and execute.

Then Run ahead execution in future execution one core of a CMP to prefetch data for a thread running on another core. The original unmodified program executes on the first core, the prefetching core simply executes a copy of all non-control instructions after they have executed in the primary core, As each instruction commits on the way to the second core, it updates the value predictor with its current result, and it's output is replaced by a prediction of the likely output that the nth future instance of this instruction will produce. After that, the committed instruction is sent to the second core along with the predicted value, where it is injected into the dispatch stage of the pipeline since it has decoded in the regular core. Instructions are injected in the commit order in the first core to preserve the program semantics. According to the spatial locality, FE assumes that the same sequence of instructions will execute again in the future, the second core essentially executes n "iterations" utilizing the future values ahead of the non-speculative program running in the first core. The instructions speculatively executed on the second core issue load requests the main program will probably reference in the future. Figure 2 shows the architecture

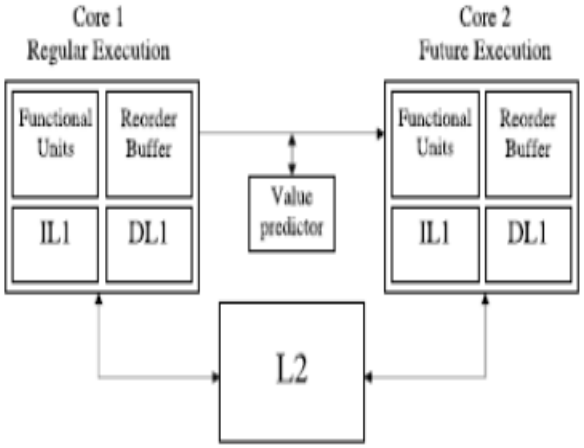

Fig. 2. The FE architecture

Then Dual-Core Execution is introduced

1) Principle: Dual-core execution consists of two superscalar cores (a front and back core) coupled with a queue. The front core fetches an instruction stream in order and executes instructions in its normal manner except for those load instructions resulting in a long-latency cache miss. An invalid value is used as the fetched data to avoid the cache-missing load blocking the pipeline, similar to the run-ahead execution. When instructions retire (in order) from the front core, they are inserted into the result queue and will not update the memory.

The second superscalar core, called the back processor, consumes the preprocessed instruction stream from the result queue and provides the precise program state (i.e., the architectural register file, program counter, and memory state) at its retirement stage. In DCE, the front core benefits the back processor in two major ways: (1)a highly accurate and continuous instruction stream as the front core resolves most branch mispredictions during its preprocessing, and (2) the warmed up data caches as the cache misses initiated by the front processor become prefetches for the back processor.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

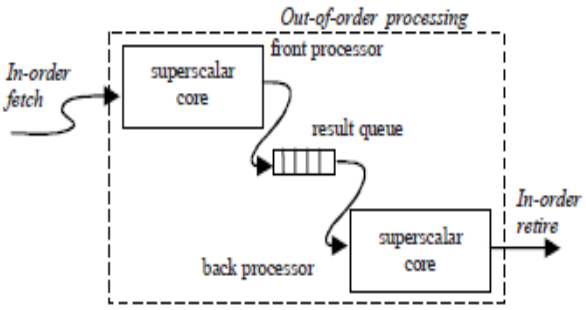

Fig. 3. The DCE architecture

DCE achieves higher performance compared to run-ahead mechanism and future Execution, for the pipeline will not stall when L2 cache miss occurs and DCE could avoid invalid prefetching. DCE provides an interesting non-uniform way to handle branches. The branches that depend on short latency operation are resolved promptly at the front core while only the branches depending on cache miss are deferred to the back core. Though DCE improves IPC, the efficiency of CPU is decreased, all instruction executes twice except for loads and branches and most instruction executed on two cores gets same result.

C. Energy-Efficient Hardware Prefetching for CMPs using Heterogeneous Interconnects.

In the last years high performance processor designs have evolved toward Chip-Multiprocessor (CMP) architectures that implement multiple processing cores on a single die. As the number of cores inside a CMP increases, the on-chip interconnection network will have significant impact on both overall performance and power consumption. CMP designs are likely to be equipped with latency hiding techniques like hardware prefetching in order to reduce the negative impact on performance that, otherwise, high cache miss rates would lead to. This method shows how to reduce the impact of prefetching techniques in terms of power (and energy) consumption in the context of tiled CMPs. This proposal is based on the fact that the wires used in the on-chip interconnection network can be designed with varying latency, bandwidth and power characteristics. By using a heterogeneous interconnect, where low-power wires are used for dealing with prefetched lines, significant energy savings can be obtained.

One of the greatest bottlenecks to provide high performance and energy efficiency in such tiled CMP architectures is the high cost of on-chip communication through global wires, wires pose major performance and power consumption problems as technology shrinks and total die area increases.

A tiled CMP architecture consists of a number of replicated tiles connected over a switched direct network. Each tile contains a processing core with primary caches (both instruction and data caches), a slice of the L2 cache, and a connection to the on-chip network. The L2 cache is shared among the different processing cores, but it is physically distributed between them. Therefore, some accesses to the L2 cache will be sent to the local slice while the rest will be serviced by remote slices. In addition, the L2 cache stores the directory information needed to ensure coherence between the L1 caches. On a L1 cache miss, a request is sent down to the appropriate tile where further protocol actions are initiated based on that block's directory state, such as invalidation messages, intervention messages, data write back, data block transfers, etc.

They propose to use two wire implementations apart from baseline wires (B-Wires): power optimized wires (PW-Wires) that have fewer and smaller repeaters, and bandwidth optimized wires (L-Wires) with higher widths and spacing. Then, coherence messages are mapped to the appropriate set of wires taking into account, among others, their latency and bandwidth requirements. There are a variety of message types traveling on the interconnect of a CMP, each one with properties that are clearly distinct. In general, we can classify messages into the following groups: *Request* messages, that are generated by cache controllers in response to L1 cache misses, or a likely future L1 cache miss when prefetching is considered, and sent to the corresponding home L2 cache to demand privileges over a memory line. Response messages to these requests, generated by the home L2 cache controller or, alternatively, by the remote L1 cache that has the single valid copy of the data, and they can carry the memory line or not. Coherence commands, that are sent by the home L2 cache controller to the corresponding L1 caches to ensure coherence. Coherence responses sent by the L1 caches back to the corresponding home L2 in response to coherence commands. Replacement messages that the L1 caches generate in case of exclusive or modified lines being replaced (replacement hints are not sent for lines in shared state).

Interconnect Design for Efficient Message Management: PW-Wires have the same area cost than baseline wires while they are twice slower. Fixing the number of PW-Wires is not a naive task. They will be used for sending prefeching-related messages (in particular, prefetch replies with data), whereas the remaining area will be consumed by B-Wires employed for sending ordinary messages and short prefeching-related messages. The proportion between PW- and B-Wires has a direct impact in both the execution time and the power consumption of the interconnect.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 8, October 2012

By tuning wire width and spacing, it is possible to design wires with varying latency and bandwidth properties. Similarly, by tuning repeater size and spacing, it is possible to design wires with varying latency and energy properties. Results obtained through detailed simulations of a 16-core CMP show that the proposed on-chip message management mechanism can reduce the power consumed by the links of the interconnection network about 23% with degradation in execution time of 2%. Finally, these reductions translate into overall CMP savings of up to 10% (4% on average) when the consumed energy is considered.

D. Energy-Efficient Hardware Data Prefetching.

This paper presents a set of new energy-aware techniques to overcome prefetching energy. These include compiler-assisted and hardware-based energy-aware techniques and a new power-aware prefetch engine that can reduce hardware prefetching related energy consumption by 7–11X . Most of the energy overhead due to hardware prefetching comes from prefetch-hardware-related energy cost and unnecessary L1 data cache lookups related to prefetches that hit in the L1 cache. Hardware-based prefetching mechanisms need additional components for prefetching data based on access patterns. Prefetch tables are used to remember recent load instructions and relations between load instructions are set up. These relations are used to predict future (potential) load addresses from where data can be prefetched. Hardware-based prefetching techniques studied in this paper include sequential prefetching

Sequential Prefetching: Sequential prefetching schemes are based on the One Block Look ahead (OBL) approach; a prefetch for block b+1 is initiated when block b is accessed. OBL implementations differ based on what type of access to block initiates the prefetch of b+1. Prefetch-on-miss sequential algorithm initiates a prefetch for block b+1 whenever an access for block results in a cache miss. If b+1 is already cached, no memory access is initiated.

Stride Prefetching: Stride prefetching monitors memory access patterns in the processor to detect constant-stride array references originating from loop structures. This is normally accomplished by comparing successive addresses used by memory instructions. Since stride prefetching requires the previous address used by a memory instruction to be stored along with the last detected stride, a hardware table called the Reference Prediction Table (RPT), is added to hold the information for the most recently used load instructions. Each RPT entry contains the PC address of the load instruction, the memory address previously accessed by the instruction, a stride value for those entries that have established a stride, and a state field used to control the actual prefetching.

C: Pointer Prefetching: One scheme for hardware-based prefetching on pointer structures is dependence-based prefetching that detects dependencies between load instructions rather than establishing reference patterns for single instructions. Dependence-based prefetching uses two hardware tables. The correlation table (CT) is responsible for storing dependence information. Each correlation represents dependence between a load instruction that produces an address (producer) and a subsequent load that uses that address (consumer). The potential producer window (PPW) records the most recent loaded values
and the corresponding instructions. When a load commits, its base address value is checked against the entries in the PPW, with a correlation created on a match. This correlation is added to the CT.

D. Combined Stride and Pointer Prefetching: a combined technique that integrates stride prefetching and pointer prefetching was implemented and evaluated. The combined technique performs consistently better than the individual techniques on two benchmark suites with different characteristics.

III. CONCLUSION

This survey concludes that, hardware prefetching of data is done in a reliable and fast way using different schemes such as multi-order analysis and multi-core hardware based on pre-execution. Along with this there exists some methodology to carry out these prefetching operations in an energy efficient manner by using techniques, such as, complier assistance and low power consumption wires.

REFERENCES

[1] Yong Chen, Huaiyu Zhu, Hui Jin, and Xian-He Sun, "Improving the Effectiveness of Context-based Prefetching with Multi-order Analysis", 39th International Conference on Parallel Processing Workshops (ICPPW), ISSN: 1530-2016, E-ISBN: 978-0-7695-4157-0, 2010, pp. 428 – 435.

[2] Juan Fang and Hongbo Zhang,"Analysis and improvement of dynamic multi-core hardware prefetch technology based on pre-execution", Fifth International Conference on Frontier of Computer Science and Technology, E-ISBN: 978-0-7695-4139-6, 2010, pp. 387-391.

[3] Antonio Flores, Juan L. Arag´on and Manuel E. Acacio, "Energy-Efficient Hardware Prefetching for CMPs using Heterogeneous Interconnects" IEEE 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, ISSN: 1066-6192 2010, pp. 147-154.

[4] Yao Guo, Pritish Narayanan, Mahmoud Abdullah Bennas, Saurabh Chheda, and Csaba Andras Moritz, "Energy-Efficient Hardware Data Prefetching", IEEE

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 19(2), February 2011,pp. 250-263.

[5] Byna S., Yong Chen, Xian-He Sun, "A Taxonomy of Data Prefetching Mechanisms**",** International Symposium on Parallel Architectures, Algorithms, and Networks, ISSN: 1087-4089, 2008, pp. 19 - 24.

[6] Srinath S., Mutlu O., Hyesoon Kim, Patt Y.N., "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers", IEEE 13th International Symposium on High Performance Computer Architecture, E-ISBN: 1-4244-0805-9, 2007, pp. 63 - 74.

[7] Nesbit K.J., Smith, J.E., "Data Cache Prefetching Using a Global History Buffer", IEE Proceedings- Software, ISSN: 1530-0897, 2004, p. 96.