Global-aware and multi-order context-based prefetching for high-performance processors

Yong Chen, Huaiyu Zhu, Philip C. Roth, Hui Jin and Xian-He Sun

International Journal of High Performance Computing Applications 2011 25: 355 originally published online 8 March 2011
DOI: 10.1177/1094342010394386

Special Issue article

THE INTERNATIONAL JOURNAL of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
25(4) 355–370
© The Author(s) 2011
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/1094342010394386
hpc.sagepub.com
SAGE

Global-aware and multi-order context-based prefetching for high-performance processors

Yong Chen[1], Huaiyu Zhu[2], Philip C. Roth[3], Hui Jin[4] and Xian-He Sun[4]

Abstract

Data prefetching is widely used in high-end computing systems to accelerate data accesses and to bridge the increasing performance gap between processor and memory. Context-based prefetching has become a primary focus of study in recent years due to its general applicability. However, current context-based prefetchers only adopt the context analysis of a single order, which suffers from low prefetching coverage and thus limits the overall prefetching effectiveness. Also, existing approaches usually consider the context of the address stream from a single instruction but not the context of the address stream from all instructions, which further limits the context-based prefetching effectiveness. In this study, we propose a new context-based prefetcher called the Global-aware and Multi-order Context-based (GMC) prefetcher. The GMC prefetcher uses multi-order, local and global context analysis to increase prefetching coverage while maintaining prefetching accuracy. In extensive simulation testing of the SPEC-CPU2006 benchmarks with an enhanced CMP\$im simulator, the proposed GMC prefetcher was shown to outperform existing prefetchers and to reduce the data-access latency effectively. The average Instructions Per Cycle (IPC) improvement of SPEC CINT2006 and CFP2006 benchmarks with GMC prefetching was over 55% and 44% respectively.

Keywords

prefetching, context-based prefetching, prefetching accuracy, prefetching coverage, processor architectures, memory hierarchy, data access delay, prefetch degree, prefetch priority, cache pollution, bandwidth contention, SPEC-CPU2006, CMP\$im simulator, PIN, high-end computing, data intensive computing

1 Introduction

The rapid advance of semiconductor process technology means that the processor speed or the aggregate processor speed on chips with multicore/manycore architectures grows quickly and steadily. The memory speed or the data load/store performance, on the other hand, has been increasing at a snail's pace (Hennessy and Patterson, 2006). The memory speed has only increased by roughly 9% each year over the past two decades, which is significantly lower than the improvement in speed of nearly 50% per year for processor performance (Hennessy and Patterson, 2006). The unbalanced performance improvement leads to one of the significant performance bottlenecks in high-end computing, known as the "memory-wall" problem (Wulf and Mckee, 1995; McKee, 2004). Worse, unless revolutionary memory technology changes occur, the processor–memory performance gap is predicted to continue to widen in the coming years. Multiple memory hierarchies have been the primary solution to bridging the processor–memory performance gap. However, due to the limited cache capacity and highly associative structure, the large number of off-chip accesses and long data-access latency still spike the performance severely.

Data prefetching is a common technique for reducing the processor stall time on data accesses, and has been widely recognized as a critical companion technique of the memory hierarchy solution to overcoming the memory-wall issue (Wulf and Mckee, 1995; Mckee, 2004; Hennessy and Patterson, 2006). As the term indicates, the essential idea of data prefetching is to observe data referencing patterns, then speculate future references and fetch the

[1]Department of Computer Science, Texas Tech University, USA
[2]Department of Computer Science, University of Illinois at Urbana Champaign, USA
[3]Computer Science and Mathematics Division, Oak Ridge National Laboratory, USA
[4]Department of Computer Science, Illinois Institute of Technology, USA

Corresponding author:
Yong Chen, Department of Computer Science, Texas Tech University, Box 43104, Lubbock, TX 79409-3104
Email: yong.chen@ttu.edu

predicted reference data closer to the processor before the processor demands them. By overlapping computation and data accesses, data prefetching can partly overcome the limitations of cache memories, and reduce long memory access latency. Many applications, including scientific and high-end computing applications, exhibit regular and predictable access patterns, and thus data prefetching works well and can effectively improve the data-access performance (Chen and Baer, 1995; Vanderwiel and Lilja, 1997; Srinivasan et al., 2004; Hennessy and Patterson, 2006). Numerous studies have been conducted and many strategies have been proposed for data prefetching (Jouppi, 1990; Dahlgren et al., 1993; Chen and Baer, 1995; Joseph and Grunwald, 1997; Kandiraju and Sivasubramaniam, 2002; Annavaram et al., 2003; Nesbit et al., 2004; Nesbit and Smith, 2004; Srinivasan et al., 2004; Emma et al., 2005; Wenisch et al., 2005; Ceze et al., 2006a,b; Srinath et al., 2007; Chen et al., 2008; Ebrahimi et al., 2009; Somogyi et al., 2009; Bhattacharjee and Martonosi, 2010). In addition, many commercial high-performance processors have adopted data prefetching techniques to hide long data-access latency (Doweck, 2006; Hennessy and Patterson, 2006; Le et al., 2007).

Among many prefetching strategies, *context-based data prefetching* has received attention in recent years due to its general applicability and high accuracy (Geomen et al., 2001; Ramos et al., 2007, 2009; Chen et al., 2010). A context-based prefetcher 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 prefetcher can adjust the prefetching overhead and confidence. The length of the context is referred as the *order* of the context-based prefetcher. For instance, if the access strides sequence is 's_1, s_2, s_3, s_4, s_5', the context prefetching can build a state transition diagram with order-1 context, i.e. 's_1', 's_2', 's_3', etc., order-2 context, i.e. 's_1, s_2', 's_2, s_3', 's_3, s_4', etc., or order-3 context such as 's_1, s_2, s_3', 's_2, s_3, s_4', etc.

Although numerous studies have been conducted in context-based prediction and prefetching (Sazeides and Smith, 1997a,b; Goeman et al., 2001; Ramos et al., 2007, 2009), many issues remain open. We have identified two major limitations of current context-based prefetching via extensive simulation. The first limitation is that the existing context-based prefetching only supports the context analysis and prediction with a single order. Although such an approach can achieve high prefetching accuracy (the ratio of the accurate prefetches to all prefetches), our study shows that using an approach that considers only one order leads to limited prefetching coverage (the ratio of reduced misses to all misses). The ultimate goal of a data prefetching strategy is to reduce access delay; however, the performance gain of data prefetching depends on both prefetching coverage and accuracy. This limited prefetching coverage, in turn, leads to limited prefetching effectiveness. The second limitation is that the existing context-based prefetching usually only considers the context of the miss address stream from a specific instruction and ignores the context of the miss address streams from other instructions. We refer to the context of a specific instruction as the *local context*, and the context of all instructions as the *global context*. We observe that, by incorporating both local and global context analysis into context-based prefetching, the prefetching coverage can be further improved. Motivated by these limitations of existing context-based prefetchers, we propose a new context-based prefetcher design called the *Global-aware and Multi-order Context-based (GMC) prefetcher*. The goal of this research is to address the limitations of existing approaches and further explore the potential of the context-based prefetching.

The contribution of this work is three-fold. First, we present a comprehensive study that compares different orders of context-based prefetching models, as well as local and global context analysis. Second, we propose a new Global-aware and Multi-order Context-based data prefetcher that incorporates multi-order, local and global context analysis to achieve better overall prefetching effectiveness. Third, we present simulation results that validate the GMC prefetcher design, showing significant performance improvement over existing data prefetching techniques.

The rest of this paper is organized as follows: Section 2 reviews important related works; Section 3 introduces the design of the proposed Global-aware and Multi-order Context-based prefetcher and prefetching methodology; Section 4 discusses the simulation experiments and performance results in detail; Section 5 concludes this study.

2 Related work

Hardware data prefetching has been extensively studied (Jouppi, 1990; Dahlgren et al., 1993; Palacharla and Kessler, 1994; Chen and Baer, 1995; Joseph and Grunwald, 1997; Kandiraju and Sivasubramaniam, 2002; Annavaram et al., 2003; Nesbit et al., 2004; Nesbit and Smith, 2004, 2005; Wenisch et al., 2005; Ceze et al., 2006a,b; Chen et al., 2007; Le et al., 2007; Srinath et al., 2007; Ebrahimi et al., 2009; Somogyi et al., 2009; Bhattacharjee and Martonosi, 2010; Chen et al., 2010). Existing representative studies can be roughly classified into several categories: sequential, stream, and stride prefetching; correlation-based prefetching; and context-based prefetching. After describing previous work in each of these categories, we discuss how our GMC prefetcher is related to the existing work.

2.1 Sequential, stream, and stride prefetching

Sequential prefetching is a basic prefetching strategy. It prefetches one or more blocks that follow the current missing block (Dahlgren et al., 1993, 1995). This prefetching mechanism takes advantage of spatial locality and assumes the applications usually request consecutive memory blocks. The One-Block-Lookahead (OBL) approach prefetches one block that follows the one being accessed. An alternative to OBL is prefetching *k* blocks of data instead of one; *k* is called the *prefetching degree*. To improve prefetching efficiency, adaptation of the prefetching degree at runtime was introduced to sequential prefetching (Dahlgren et al., 1993). Stream prefetching can track different streams and prefetch data blocks for each stream (Jouppi, 1990; Palacharla and Kessler, 1994). It is adopted in commercial processors such as POWER6 architecture processors (Le et al., 2007). Stride prefetching was proposed by Chen and Baer to detect the stride patterns in data access streams (Chen and Baer, 1995). A Reference Prediction Table (RPT) is used to keep track of the strides of recent data accesses. Once the prefetcher is trained, blocks can be prefetched according to the stride recorded in the RPT. Due to its simplicity and effectiveness, stride prefetching is widely used (Chen and Baer, 1995; Doweck, 2006).

2.2 Correlation-based prefetching

Markov prefetching was proposed by Joseph and Grunwald to capture the correlation between cache misses and prefetch data based on a state transition diagram (Joseph and Grunwald, 1997). Nesbit and Smith proposed the Global History Buffer (GHB) to maintain the recent history of data access (Nesbit and Smith, 2004, 2005). It is an efficient structure for supporting different prefetching algorithms and has been adopted widely. Similar to GHB, the Data Access History Cache (DAHC) (Chen et al., 2007) was proposed to facilitate data prefetching with a single structure and to support various algorithms simultaneously at runtime. More recently, a stream chaining method was proposed by Diaz and Cintra to link various localized streams into predictable chains such that multiple levels of correlation can be exploited by the prefetcher (Diaz and Cintra, 2009). Since data access patterns may change at runtime, adaptive mechanisms, such as the feedback directed prefetching proposed by Srinath et al. (2007), are often used to control dynamically the prefetch degree and the prefetcher's aggressiveness.

In addition, epoch-based correlation prefetching was proposed by Chou to reduce late prefetches for correlation-based prefetching (Chou, 2007). The idea is to localize misses within an epoch (a fixed length of time), and to conduct prefetches based on epochs to prevent late prefetches. Lai et al. (2001) proposed a dead-block predictor (DBP) to improve the timeliness of correlation-based prefetching by triggering prefetches and making replacement decisions on time. It predicts when a cache block is dead and can be evicted by tracking the time duration between when it comes into the cache and when it is evicted. Hu et al. (2002) also proposed time-keeping mechanisms to improve timely prefetching. Bhattacharjee and Martonosi proposed two Inter-Core Cooperative (ICC) Translation Lookaside Buffer (TLB) prefetching mechanisms to exploit the correlation in TLB miss patterns across cores in chip multiprocessors (CMPs), which significantly improves data TLB (D-TLB) access performance (Bhattacharjee and Martonosi, 2010). Ebrahimi et al. (2009) proposed a hierarchy of prefetcher aggressiveness control structures to control prefetcher-caused inter-core interference by dynamically adjusting and coordinating the aggressiveness of multiple prefetchers in CMPs. Their proposed structures improve system performance and reduce bus traffic considerably on a multi-core system. Somogyi, Wenisch, et al. proposed temporal, spatial, and spatio-temporal memory streaming to detect repeated patterns in specific memory regions and to boost memory access performance (Wenisch et al., 2005; Somogyi et al., 2009). These approaches achieve remarkable performance for applications with regional repeated patterns.

2.3 Context-based prefetching

Context-based prefetching is considered to be a more generalized prefetching approach than conventional prefetching strategies. In context-based data prefetching, the correlation between the current context (the miss access) and the past history is used to make predictions for data prefetching. A context-based prefetching method builds a state transition diagram with the access address strides (deltas) as states, and characterizes the correlation among miss address streams.

Context-based data prefetching is similar to the context-based value predictor proposed by Sazeides and Smith (1997a,b), but is used as a data prefetcher at the cache level. The Finite Context Method (FCM) is a representative context-based predictor that predicts the next value based on a finite number of preceding values (Sazeides and Smith, 1997b). The FCM is usually implemented with two tables (Sazeides and Smith, 1997a), a Value History Table (VHT) and a Value Prediction Table (VPT). The VHT contains the context of data accesses, and the VPT contains the predictions associated with the context. A hash function uses the context information from the VHT and the processor state to form an index that leads to a VPT entry. A variation of the FCM prefetcher, called Differential Finite Context Method (DFCM), was proposed by Goeman et al. (2001). In this model, the context is formed by the differences between values instead of the values themselves. DFCM can find more repeating patterns than FCM does, using fewer VPT entries than FCM requires. P-DFCM is another proposed data prefetcher based on DFCM (Ramos et al., 2007). There are two major differences between P-DFCM and DFCM. First, P-DFCM prefetches on L2 load misses only, while DFCM prefetches on both load and store misses. Second, P-DFCM requires only the program counter (PC) to prefetch data, whereas DFCM requires both the PC and the requested data address. Distance prefetching, originally proposed for TLB but also used in data caches, is usually considered a representative context-based prefetcher with order-1 context analysis (Kandiraju and Sivasubramaniam, 2002). Recent extensions of distance prefetching, called PC/DC and G/DC prefetchers (Nesbit et al., 2004; Nesbit and Smith, 2005), improve the distance prefetching and are representative order-1 prefetchers.

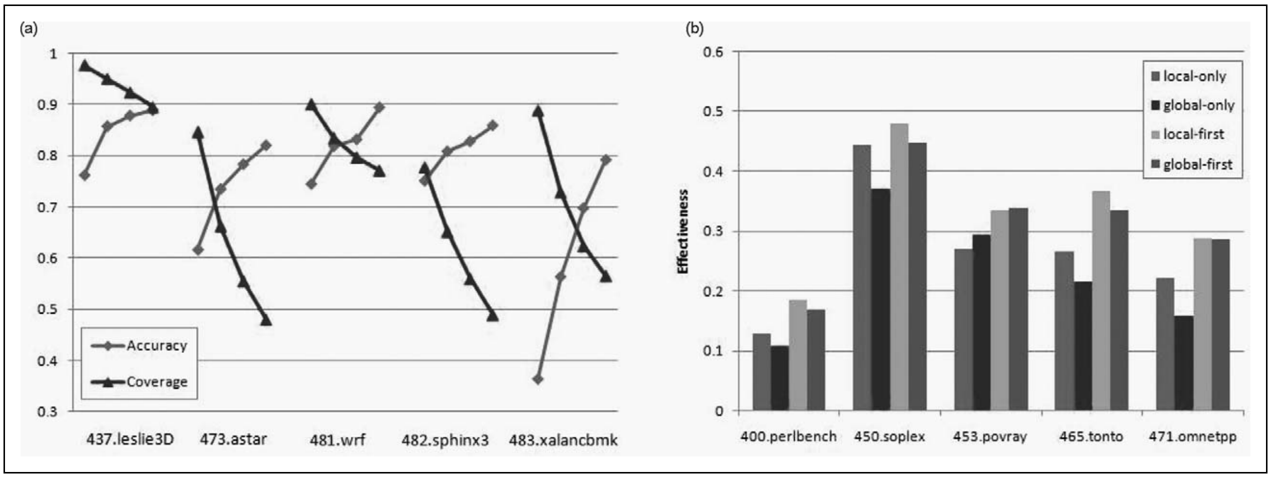

Figure 1. Impact of orders and context scope on context-based prefetching performance.
(a) Impact of different orders with representative SPEC-CPU2006 benchmarks (b) Impact of different context scope with representative SPEC-CPU2006 benchmarks

2.4 Relation to this work

In this paper, we propose a new Global-aware and Multi-order Context-based prefetcher. We also present a comprehensive evaluation of the performance of context-based prefetching. We analyze the impact of orders, local context and global context on the performance. To the best of our knowledge, there does not exist a detailed study that analyzes context-based prefetching performance and the impact of different orders and contexts across a wide range of prefetching techniques. In addition, existing context-based prefetchers only adopt the context analysis with a single order, and do not take both local context and global context into consideration at the same time. Our evaluation shows that these drawbacks of existing strategies lead to limited prefetching coverage, and in turn limit the overall effectiveness of prefetching. The proposed GMC prefetcher addresses these limitations well. The simulation results show that this new prefetcher outperforms the existing context-based prefetchers and achieves considerably better latency reduction than the existing prefetchers.

3 GMC prefetcher design and prefetching methodology

In this section we present the design and prefetching methodology of the Global-aware and Multi-order Context-based (GMC) prefetcher. We first introduce the observations that motivate the proposed GMC prefetcher. We then introduce the design rationale and a high-level view of the prefetcher. After that, we detail the GMC prefetcher techniques including prefetching methodology, table updating, prefetch priority and degree control. Lastly, we discuss the implementation cost of a GMC prefetcher.

3.1 Motivation

The motivation of the proposed GMC data prefetcher comes from two important observations we noticed in evaluating the performance of context-based prefetching. The performance evaluation was conducted from two aspects: the *prefetching accuracy*, which measures what percentage of prefetches are correct, and the *prefetching coverage*, which measures how many patterns are recognized and how many misses are reduced. We used another metric, *prefetching effectiveness*, to evaluate the performance of a prefetching strategy considering both accuracy and coverage. Prefetching effectiveness is defined as the product of the prefetching accuracy and the prefetching coverage. This definition of the prefetching effectiveness metric is motivated by the common usage of an aggregated speedup metric (Hennessy and Patterson, 2006); accuracy and coverage are two independent metrics that evaluate a prefetching strategy, while the effectiveness is an aggregation of these two metrics.

Our first observation was that context-based analysis with higher order can achieve better accuracy in generating predictions than analysis with lower order. For instance, order-2 analysis has higher accuracy than order-1 analysis. However, using an approach that considers only one order (e.g. order 2 but not order 1 or order 3) limits the recognition rate of patterns and thus the prefetching coverage. This limited prefetching coverage in turn also affects the prefetching effectiveness. Figure 1(a) shows the simulation results of the prefetching accuracy and coverage of five representative SPEC-CPU2006 benchmarks (Section 4 has the results of all 29 benchmarks) with orders 1, 2, 3, and 4 respectively from left to right in each group. The results show that a high-order prefetcher has higher accuracy but lower coverage, and a low-order prefetcher has lower accuracy but can considerably improve the prefetching coverage. This observation motivates us to integrate multi-order analysis to provide both high accuracy and wide coverage for context-based data prefetching.

The second observation is that the existing context-based prefetchers only consider the context scope of access addresses from the same instruction, because this approach

Table 1. Four different strategies considering context scope analysis.

Strategy	Description
Local-only	Adopt local context analysis only; ignore global context analysis
Global-only	Adopt global context analysis only; ignore local context analysis
Local-first	Adopt both local and global context analysis, with a higher priority for the local context analysis
Global-first	Adopt both local and global context analysis, with a higher priority for the global context analysis

has high accuracy in identifying potential correlations. However, the simulation testing shows that this approach has low overall prefetching effectiveness. When considering the context scope analysis, we distinguish four strategies: *local-only*, *global-only*, *local-first*, and *global-first*. The *local-only* context scope considers a strategy that only uses the local context analysis, while ignoring any global context analysis. Similarly, the *global-only* strategy only uses the global context analysis. The *local-first* context scope analysis refers to a strategy that combines both local and global context analysis but always gives the local context analysis a higher priority. The *global-first* strategy uses both local and global context analysis, but gives global context analysis a higher priority. The differences between these four strategies are also shown in Table 1 for easy comparison.

Figure 1(b) illustrates the prefetching effectiveness of an order-2 context-based prefetcher with five representative SPEC-CPU2006 benchmarks. The results clearly demonstrate the potential of global context awareness. Among the four strategies, the local-first global-aware strategy achieved the best prefetching effectiveness. This observation motivates us to incorporate global awareness into context-based data prefetching.

3.2 GMC prefetcher design and rationale

Motivated by the observations, we propose a new Global-aware and Multi-order Context-based data prefetcher to incorporate multi-order, local and global context analysis to achieve better overall prefetching effectiveness. The proposed GMC prefetcher has a three-level table organization as shown in Figure 2. The first two levels adopt the design of the GHB prefetcher with separate PC and address indices (Nesbit and Smith, 2004, 2005). The GHB table is a FIFO structure and stores detailed data access history information. Each entry in the GHB table is a pair containing a program counter and access address. It gives high priority to recent access history, and thus removes the most outdated history automatically. The indices are used to provide quick access to the GHB table with a "key". The unique instruction address is the key for program counter indices, and the unique data address is the key for address indices. The

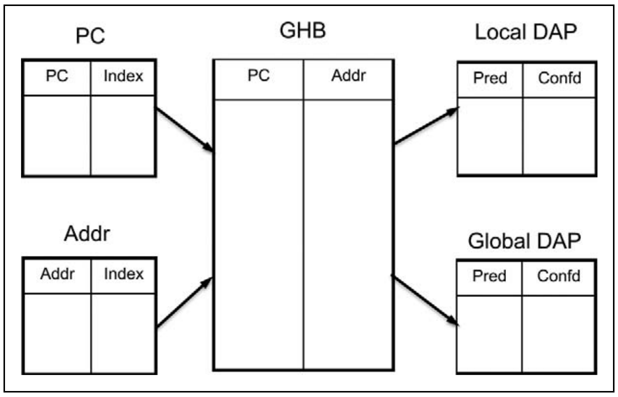

Figure 2. High-level view of GMC prefetcher design.

content of the PC and address index tables are the paired PC and index, and address and index, respectively. GMC augments GHB with a third level of tables – the Data Access Prediction (DAP) tables (see Figure 2). There are two DAP tables: Local DAP and Global DAP. The Local DAP table is used to store the prediction associated with the address context coming from the same PC (local context). The Global DAP table is used to store the prediction associated with the address context coming from any PC (global context). 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 our current design, we use an FS-5 hash function to produce the hashed context as it has better performance than other choices (Sazeides and Smith, 1997a). In the remainder of this section, we present the detailed design and the methodology of supporting context-based prefetching based on local, global and multi-order context analysis in the GMC prefetcher respectively.

3.3 Local context prefetching methodology

Figure 3 shows the methodology of prefetching based on local context analysis. When a new miss occurs, the prefetcher searches the PC indices 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. The recent k strides (order-k context) are fed into the hash function unit and a hashed form of context is the output. The GMC prefetcher uses the strides between miss addresses as the context instead of using the addresses themselves. This approach is the same with existing context-based prefetchers such as DFCM (Ramos et al., 2007). The hashed context in our current design consists of 9 bits. It is used as an index for looking up an entry in the local DAP table. If an entry can be found in the local DAP table with the predicted stride in its prediction field, prefetch requests are generated and issued. The confidence counter associated with each DAP entry reflects the confidence of the prediction following a

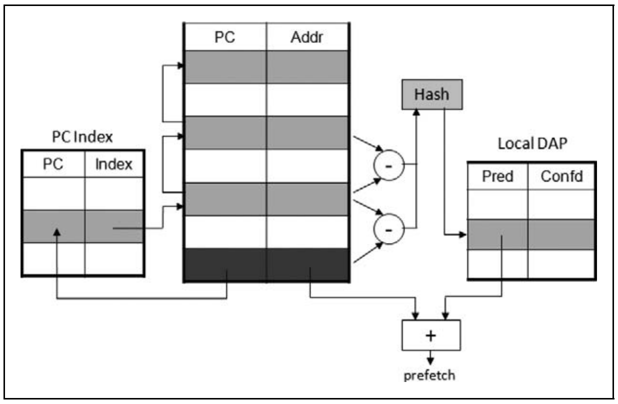

Figure 3. GMC local context prefetching.

certain context. The confidence counter (2 bits) allows four values (0, 1, 2, and 3) indicating lowest, low, high, and highest confidences. The counter value increases upon a correct prediction and decreases upon a false prediction. The confidence counter affects the decision of replacing an entry in the DAP table and issuing a prefetch request. Its usage in replacing entries is discussed later in the table updating section (Section 3.6). When the confidence counter indicates a high or highest confidence, i.e. with value 2 or 3, a prefetch is generated and the prefetch address is calculated as the current miss address plus the predicted stride.

3.4 Global context prefetching methodology

The main difference between global context prefetching and local context prefetching in the GMC prefetcher is due to the way the context is obtained. Local context prefetching considers the miss address stream from a single instruction, while global context prefetching considers the miss address stream from all instructions. Therefore, as illustrated in Figure 4, when a new miss occurs, *k* number of previous misses stored in the GHB are used with the current one to form a global context sequence. The prefetcher calculates the strides between adjacent miss addresses in this sequence, and *k* strides are fed into the hash function unit to generate the hashed form of the context. The hashed context is used to look up an entry in the global DAP table. Similar to local context prefetching, a prefetch is issued if an entry is found in the global DAP table and if its confidence is at high levels (a 2 or 3). If a prefetch is to be generated, its address is calculated as the current miss address plus the predicted stride.

3.5 Multi-order context prefetching methodology

The GMC prefetcher is designed to support multi-order analysis in context-based prefetching in order to guarantee both high prediction accuracy and wide coverage. However, we have not yet addressed the issue of *which multiple orders* should be used. The major constraint for the selection of a specific order comes from the hardware storage requirement for supporting the context analysis

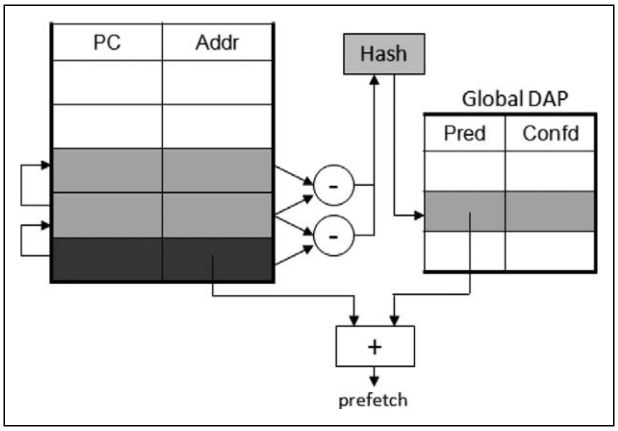

Figure 4. GMC global context prefetching.

with a specific order. The storage requirement, in turn, can be characterized by the table entry consumption for storing the context and prediction addresses in an ideal case without any hash conflicts. In order to better understand the choice of orders, we show the table size requirements for different orders in the ideal hashing case in Figure 5.

These results show that the higher the order, the more table entries we need. This is expected because the higher-order analysis uses a longer context for pattern recognition and prediction. Also, since we use a hash table (with the FS-5 hash function) in the context analysis, a higher-order context analysis can cause more aliasing for a given table size. Because higher orders require more storage or increase the likelihood of hashing conflicts, we limit GMC to use at most order 2 in our current design.

The GMC prefetcher supports order-0 and order-1 prefetching directly by taking advantage of the GHB table without involving additional DAP tables. The order-0 context prefetching means that the prefetcher works without a context. This prefetching method is essentially a sequential prefetcher, which is optimal for programs that access

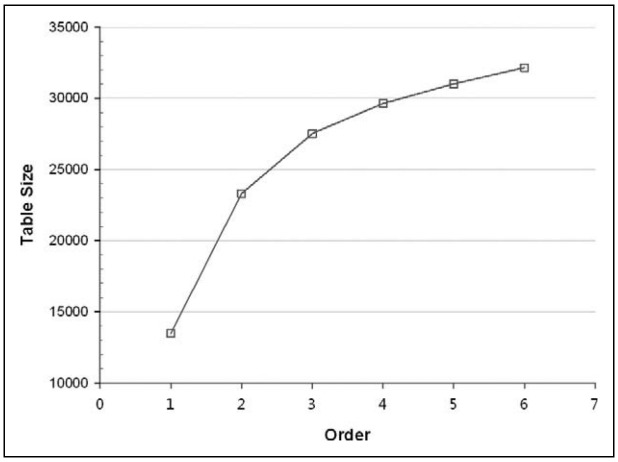

Figure 5. Table entry consumption for different orders in an ideal case.

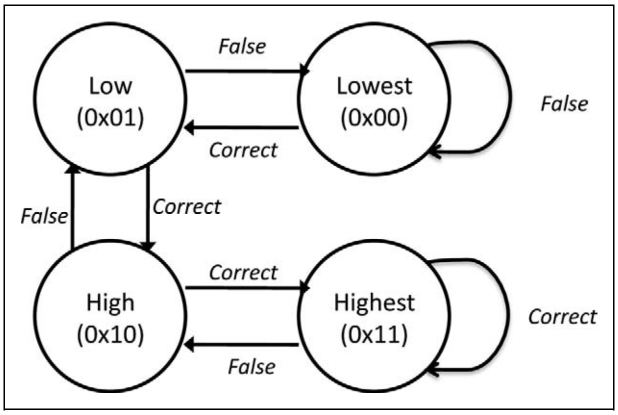

Figure 6. State transition diagram of the confidence counter.

consecutive memory blocks. The order-1 context prefetching means that the prefetcher makes the prediction based on the context with each stride as one state, which is essentially the existing distance prefetcher (Kandiraju and Sivasubramaniam, 2002; Nesbit et al., 2004). In theory, multiple DAP tables are needed so that different order contexts can be detected by the prefetcher, requiring higher hardware investment than our proposed GMC prefetcher. However, our prefetcher can carry out low-order context prefetching (order 0 and order 1) without additional DAP tables. This is because the GHB table maintains the full information for recent data accesses. The GMC prefetcher obtains the order-0 analysis (past accesses) and order-1 analysis (strides among accesses) directly from the GHB table. Order-2 prefetching requires information from the local and global DAP tables as described in the previous section. One key advantage of the GMC prefetcher is that it covers both sequential and distance prefetching and harmonizes them with the multi-order analysis.

In the GMC prefetcher, multi-order context-based prefetching operates as follows. Upon a cache miss, the order-0 prefetching issues n blocks of data requests, depending on the prefetching degree, following the miss address. The order-1 prefetching is similar to the distance prefetching based on GHB nesm04, (Nesbit and Smith, 2004, 2005). The order-2 prefetching utilizes the local and global DAP tables and follows the mechanism discussed in the previous two subsections. The prefetch requests generated from the different order analyses are handled with prefetch priority and degree control, as discussed in Section 3.7.

3.6 Table updating

The GMC hardware tables are updated when a miss occurs and before the GMC prefetcher generates prefetches. When a new miss occurs, the GMC prefetcher finds a new entry in the GHB table, replacing the oldest entry following the FIFO policy. The corresponding entries in the index tables are also updated. For the DAP tables, the previous miss address is subtracted from the new miss address and a stride Δ_1 is obtained. Then the previous $k+1$ addresses (with order-k analysis) are used to calculate the hashed form of the stride context. This hashed value is then used as an index in locating local/global DAP tables. If an entry is found and the predicted stride in that entry Δ_2 is equal to Δ_1 (which means the prediction is correct), the confidence counter is increased by one. If not, the prefetcher decides whether it should use Δ_1 to replace Δ_2 according to the confidence counter. If the confidence counter is low (with value 0 or 1), the prefetcher replaces the old stride with the new stride, and resets the confidence counter. Otherwise, the prefetcher keeps the old stride and decreases the confidence counter by 1. In essence, the confidence counter is a 2-bit saturating counter used to control the prefetcher's behavior. The state transition diagram for the confidence counter is shown in Figure 6.

3.7 Prefetch priority and degree control

To handle the prefetch requests generated from different context analysis and to guarantee the efficiency of prefetching, the GMC prefetcher supports prefetch priority and adaptive prefetch degree control. The GMC prefetcher supports multiple prefetch queues. Each queue corresponds to one order analysis. The queue with the highest-order analysis has the highest priority when GMC issues prefetches, as the highest-order analysis can generate the most accurate predictions. In the GMC's local-first prefetch strategy, within each priority queue the prefetches based on local context analysis have higher priority than the prefetches based on global context analysis. Prefetches are issued from the high priority queue first, and when more issue bandwidth is available, prefetches from lower priority queues are issued. This mechanism ensures that prefetches are always issued using the most accurate predictions available. In addition, when a prefetch queue is full, new prefetch requests replace the old ones in a FIFO manner. Note that all prefetch requests have lower priority than demand requests, so prefetch requests are only generated if there is more issue bandwidth available after the demand requests are issued. This design keeps the prefetch requests from competing with demand requests for issue bandwidth.

The prefetch degree is dynamically tuned by tracking the performance trend and bandwidth contention (Ramos et al., 2009). We use Accesses Per Cycle (APC) as the performance metric, which can be measured via hardware counters available on modern commodity processors. The GMC prefetcher checks the APC periodically and either increases or decreases the degree based on the APC value. Within a fixed window of cycles, if the APC is higher than last time it was checked, more accesses can be issued. Thus, the GMC adjusts the degree the same way (increase or decrease) that it did the previous time it checked; otherwise, it adjusts the degree in the opposite direction. Note that increases in APC do not necessarily cause increases in degree—it may be that APC increases when the degree

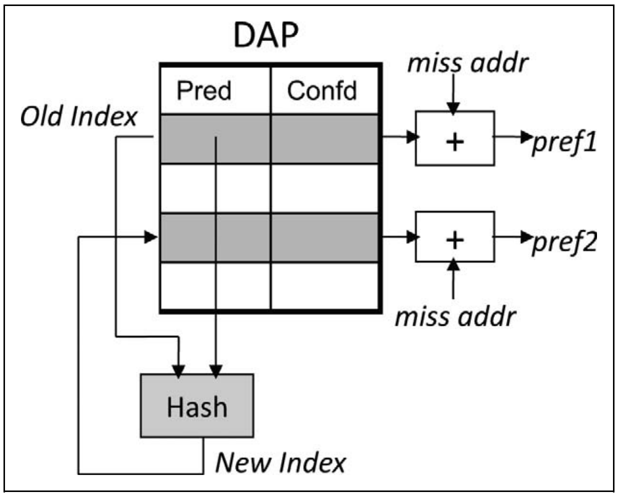

Figure 7. Multiple prefetch candidates generation.

is decreased, and our approach dynamically responds to this scenario. In addition to performance trend tracking, the GMC prefetcher also uses bandwidth contention tracking for adaptive prefetch degree control. GMC uses two groups of miss status handling registers (MSHRs) to track all on-the-fly memory access requests. One of them is used to filter the redundant misses from the stream and to hold outgoing requests issued by the CPU. The other one, called prefetch MSHRs (P-MSHRs), holds outgoing requests issued by the GMC prefetcher. In every cycle, the GMC prefetcher checks the number of requests in both MSHRs and P-MSHRs. If the number is higher than a threshold, this cycle is considered as a "busy" cycle. GMC uses the average number of requests, as analyzed through simulation testing, as a threshold. If the busy cycle number during a fixed number of cycles is higher than the threshold for high bandwidth contention, the prefetching degree is decreased even though the APC mechanism might suggest an increase. In practice, we limit the prefetching degree to the range of 1 to 32. Figure 7 shows the methodology of generating multiple prefetch candidates with prefetch degree 2 as an example. The first prefetch candidate is generated by taking the first prediction and the old index as input to the hashing unit and using the output to locate an entry in the DAP table. The second prefetch candidate is generated by combining the miss address and the prediction in the located entry. Prefetch candidates with further prefetch degree are similarly generated.

3.8 Hardware Cost

The hardware budget of the proposed GMC prefetcher is comparable with other prefetchers proposed recently (Nesbit et al., 2004; Nesbit and Smith, 2004, 2005; Diaz and Cintra, 2009). The primary hardware budget comes from the cost of hardware tables. Depending on the specific configuration, i.e. the number of entries of each table, the hardware budget ranges from 6 kB to 12 kB for 512-entry to 1024-entry tables. Compared with the well-known GHB prefetchers (Nesbit and Smith, 2004, 2005), the additional hardware budget of the GMC prefetcher comes from two additional DAP tables. Each DAP table entry needs 14 bits for the access strides and 2 bits for the confidence counter. Two 512-entry DAP tables need an additional 2 kB storage compared to GHB prefetchers. For a typical 1 MB L2 cache, the total hardware budget of 12 kB is trivial – only around 1%. However, as our simulation verifies, the global-aware and multi-order context-based prefetching can considerably reduce cache misses and improve the overall system performance, so this increase in hardware cost is well justified.

The combinatorial logic of realizing the proposed GMC prefetcher is not complicated. The primary combinatorial logic requirements are for the computation of the hashed form of context. This can be done with known and predefined functional units (Sazeides and Smith, 1997a). The rest of the GMC's required combinatorial logic supports table lookup and the computation of access strides and prefetch candidates. These operations can be implemented with simple arithmetic logic units.

Due to the increasing importance of restricting system power consumption, any proposed architectural change must consider the impact on not only hardware budget but also power. In our current study, we focus on the functionality, simulation, and analysis of the GMC prefetcher, and do not quantitatively analyze its power consumption. This approach for evaluating a new prefetcher has been widely used in existing and well-known studies (Nesbit et al., 2004; Nesbit and Smith, 2004, 2005; Doweck, 2006; Diaz and Cintra, 2009; Ramos et al., 2009; Somogyi et al., 2009), since the data prefetcher is a relatively small and independent unit. Since the storage and combinatorial logic requirements of the proposed GMC prefetcher are comparable with existing prefetchers, such as the GHB prefetcher (Nesbit and Smith, 2004, 2005) and Intel IP prefetcher (Doweck, 2006), we believe the power consumption will not become a significant barrier to the use of the GMC prefetcher, and leave a detailed quantitative analysis of GMC power consumption for future work.

4 Simulation and performance analysis

The GMC prefetcher design is general and can be used at any cache level in the memory hierarchy. In this study, we deploy it at the L2 cache level and assume this is the lowest cache level in the memory hierarchy. To evaluate the GMC prefetcher, we simulated it using the CMP$im simulator (Jaleel et al., 2008). In this section we first introduce the experimental setup, followed by a comprehensive analysis of existing context-based prefetching by evaluating the impact of different orders and the impact of local and global context scope analysis. We then present the performance results of the GMC prefetcher and a comparison of its performance with two representative prefetchers, PC/DC and G/DC (Nesbit and Smith, 2004, 2005). These prefetchers can be categorized as an order-1 local context based prefetcher and an order-1 global context based prefetcher respectively. Finally, we vary different experimental settings to test the sensitivity of the GMC prefetcher and to analyze its potential for cache pollution.

4.1 Experimental setup

We collected miss stream samples from the SPEC-CPU2006 benchmark suite and replayed them for performance analysis using Pin (Luk et al., 2005) and CMP$im (Jaleel et al., 2008). Pin is a dynamic binary instrumentation tool and can be used to collect program traces and perform various program analyses (Luk et al., 2005). CMP$im is a trace-driven simulator that characterizes memory system performance for various workloads (Jaleel et al., 2008). The first Data Prefetching Competition (DPC-1) committee released a prefetcher kit that provides an interface for integrating the CMP$im simulator with any add-on prefetching module (DPC-1, 2008). We used this prefetcher kit to simulate context-based prefetching and evaluate its performance. We used all 29 SPEC-CPU2006 benchmarks (Spradling, 2007) for our evaluation. The benchmarks were compiled using GCC 4.1.2 with -O2 optimization. For all benchmarks, we collected traces by fast-forwarding 40 billion instructions and then collecting data for the next 100 million instructions. This approach, and the evaluation setting (or a slight variation thereof), are widely used for evaluating architectural enhancements (Nesbit et al., 2004; Nesbit and Smith, 2004, 2005; Diaz and Cintra, 2009; Ramos et al., 2009; Somogyi et al., 2009). We used the *ref* input size for all benchmarks.

The simulator was configured as a representative contemporary out-of-order issue processor with a 15-stage, 4-wide pipeline. In this scenario, a maximum of two loads and a maximum of one store can be issued every cycle. The L1 cache was set as 32 kB and 8-way set associative; the L2 cache was configured as 16-way set associative, and the capacity was varied from 512 kB to 2 MB for sensitivity analysis. Both L1 and L2 caches used the LRU replacement policy. The access latency was set as 20 cycles for the L2 cache and 200 cycles for memory. The GMC prefetcher's GHB table had 1024 entries, and the index table had 512 entries. Since we used 9 bits for the hashed form of context, the GMC DAP tables had $2^9 = 512$ entries. Each entry access of the GHB, index or DAP tables was modeled as costing one cycle, which was used to calculate the prefetcher's overhead and the timing of the prefetches. This assumption of the table access overhead is reasonable for a fairly small hardware table and is also commonly used in existing studies (Nesbit et al., 2004; Nesbit and Smith, 2004, 2005; Diaz and Cintra, 2009). The PC/DC and G/DC prefetchers used the same 1024-entry GHB. When a repeated pattern was found, a fixed number of 8 prefetches were issued in the PC/DC and G/DC prefetchers. Table 2 details our simulation settings.

Table 2. Configuration of simulated processor

Parameter	Value
Window Size	128-entry
Issue Width	4
L1 Cache	32 kB, 8-way
L2 Cache	512 kB/1 MB/2 MB, 16-way
Block Size	64 B
L2 Cache Latency	20 cycles
Memory Latency	200 cycles
GHB Table Size	1024 entries
Index Table Size	512 entries
DAP Table Size	512 entries
GHB/Index/DAP Table Latency	1 cycle per entry access

4.2 Performance analysis of context-based prefetching

In this section we analyze the performance of context-based prefetching and investigate the impact of using both local and global contexts and different prefetch orders in detail.

4.2.1 Analysis of the impact of orders. In this section, we present the analysis of the impact of orders on prefetching accuracy and coverage. Figure 8 shows the accuracy of context-based prefetching with different orders for CPU2006 benchmarks. For most benchmarks (23 out of a total of 29), the accuracy increases strictly with an increase of the order, meaning that a larger order can provide higher prediction accuracy. This is intuitive because longer repeating patterns give higher prediction confidence. For the remaining six benchmarks, the accuracy does not always increase with the increase of the order but it rarely decreases. Thus, there are some application access patterns for which an increase in order (hence consideration of a longer repeating pattern) does not generate a better prediction. In addition, note that the increase ratio of the accuracy with the increase of the order differs greatly among different benchmarks.

Prefetching accuracy is important, but because there may not be enough effective prefetches to reduce cache misses and to hide memory access latency, we also care about prefetching coverage. The prefetching coverage metric is used to measure the number of patterns recognized by a prefetcher and to evaluate the percentage of misses reduced among all raw misses. As shown in Figure 9, lower prefetching orders achieve considerably wider coverage because the analysis with lower orders can recognize more patterns than the analysis with higher orders does. Hence a model with lower order can reduce more misses on the same given sequence of misses than higher-order models. This is an important observation: even though a high-order model can provide high accuracy, it may not cause a significant reduction in the number of misses, e.g. if it only issues prefetches when it is highly confident.

4.2.2 Analysis of the impact of context scope. In this section, we consider the impact of context scope (local and/or

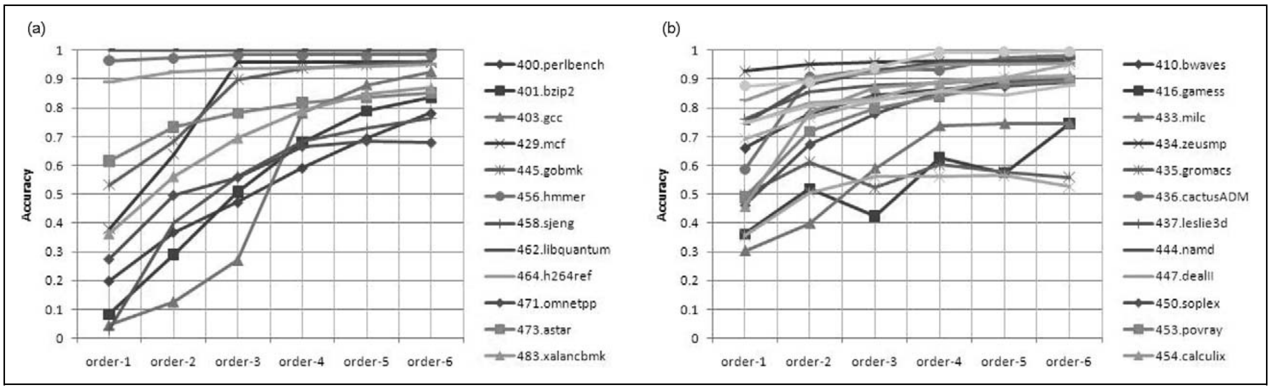

Figure 8. Prefetching accuracy of context-based prefetching with different orders for SPEC-CPU2006 benchmarks.
(a) SPEC CINT2006 benchmarks (b) SPEC CFP2006 benchmarks

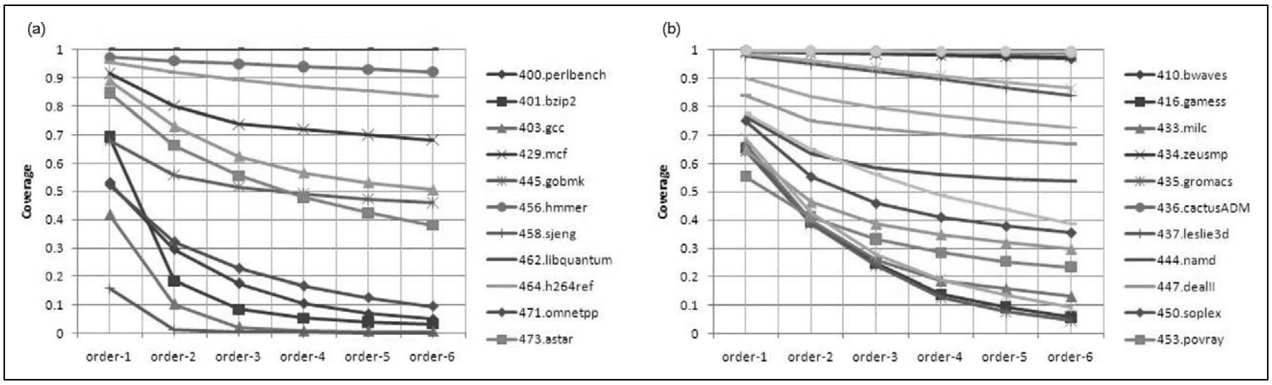

Figure 9. Prefetching coverage of context-based prefetching with different orders for SPEC-CPU2006 benchmarks.
(a) SPEC CINT2006 benchmarks (b) SPEC CFP2006 benchmarks

global) on prefetching accuracy, coverage and overall effectiveness. In this study, we restrict our presentation of the performance comparison results for these four strategies to a sample order-2 context analysis due to space limitations. The prefetching accuracy and coverage using these context analysis strategies for all 29 benchmarks are shown in Figures 10 and 11 respectively. The overall prefetching effectiveness for all benchmarks is shown in Figure 12. These results show that, although the existing and widely-adopted local-only strategy can achieve high accuracy, its prefetching coverage is not as high as the accuracy. For 27 out of 29 benchmarks, local-only prefetching coverage is lower than the strategies considering both local and global context analysis. Its overall prefetching effectiveness, as depicted in Figure 12, is also not as good. The strategies with both local and global context analysis can considerably increase the prefetching coverage and thus improve the overall effectiveness. Among the three strategies with global context analysis, the local-first strategy achieves the best overall prefetching effectiveness for the majority of benchmarks depicted in Figure 12.

The average prefetching accuracy and coverage and the overall prefetching effectiveness of four different strategies with all 29 benchmarks are shown in Figure 13. This figure has three sections, with each section showing the average result of the prefetching accuracy, the prefetching coverage and the overall prefetching effectiveness, respectively, across all 29 benchmarks. Several observations can be made from this figure. First, the approaches using local context analysis tend to achieve higher accuracy. Second, the local-first and global-first approaches can recognize more data patterns and have better prefetching coverage since they combine both local and global context analysis. Third, the local-first approach achieves the highest overall prefetching effectiveness (over 60%).

The performance trends of the context analysis with various orders, as shown in Figures 8 and 9, and with local and global scopes, as shown in Figures 12 and 13, demonstrate the need to support multiple orders and global context analysis in order to have the merits of both high accuracy and wide coverage. Furthermore, because the local-first strategy proved most effective in our evaluation, we adopt this strategy for our proposed GMC prefetcher. We present the performance analysis of the GMC prefetcher in the next subsection.

4.3 Performance analysis of GMC prefetcher

In this subsection We present the simulation results of the proposed GMC prefetcher, including the lowest-level cache (L2 cache) miss rate reduction and the IPC speedup. We also compare the GMC prefetcher's performance with

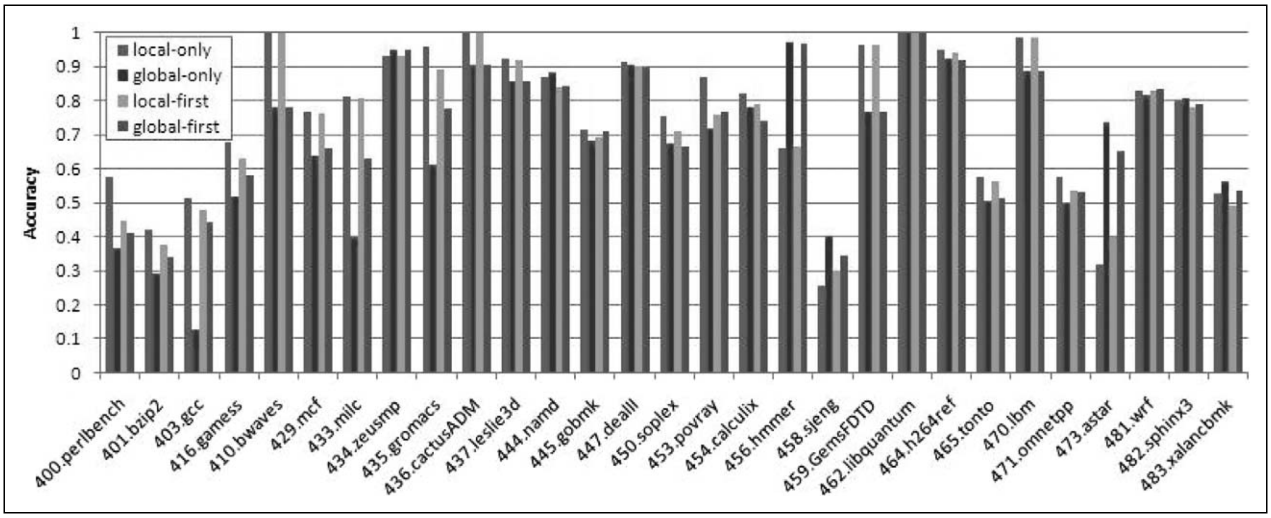

Figure 10. Prefetching accuracy of local-only, global-only, local-first and global-first strategies.

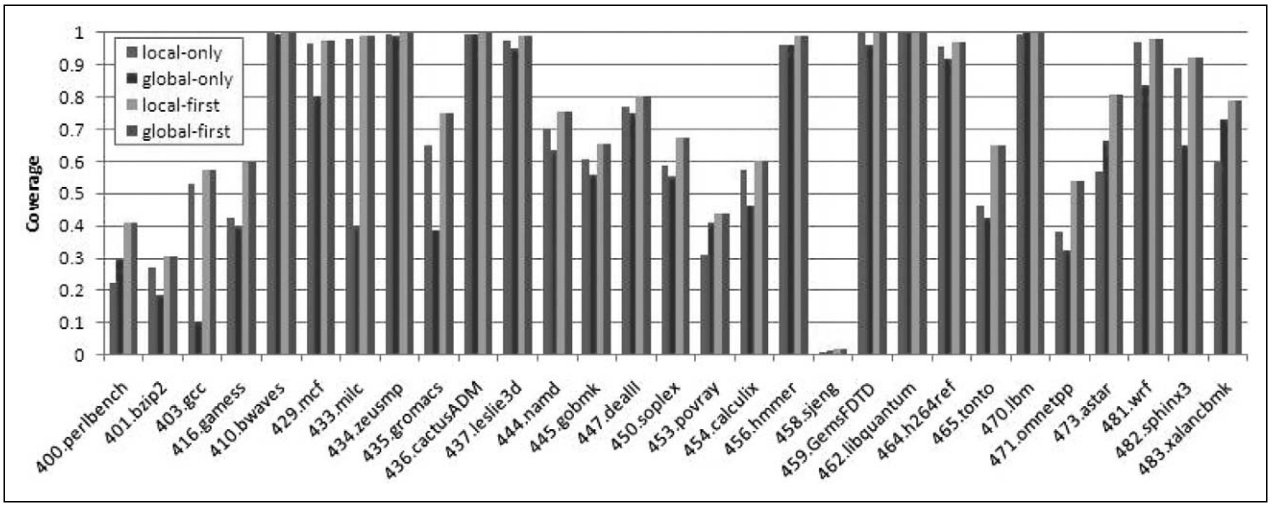

Figure 11. Prefetching coverage of local-only, global-only, local-first and global-first strategies.

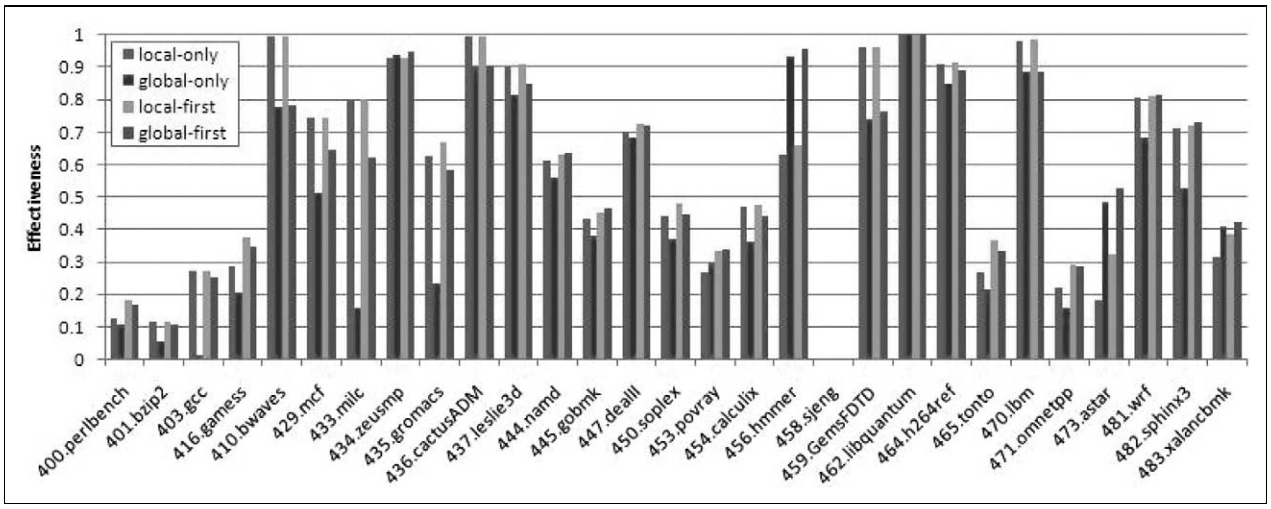

Figure 12. Overall prefetching effectiveness of local-only, global-only, local-first and global-first strategies.

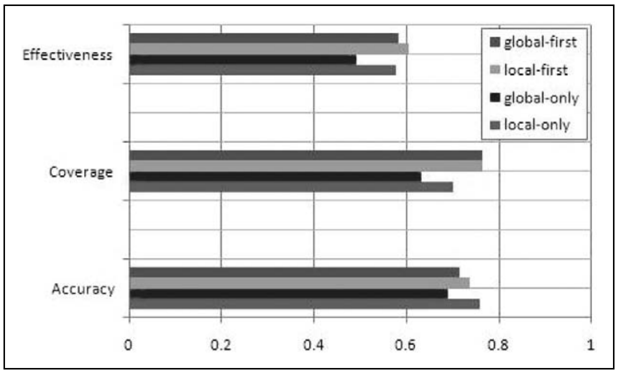

Figure 13. Average accuracy, coverage and effectiveness of local-only, global-only, local-first and global-first strategies.

two other representative and recently proposed prefetchers, PC/DC and G/DC. The L2 cache size was configured as 512 kB for these tests.

4.3.1 Miss rate reduction. Figure 14 shows the percentage of L2 cache misses reduced by the GMC prefetcher, the PC/DC prefetcher and G/DC prefetcher. On average, GMC reduced L2 cache misses by over 55% for the CINT2006 benchmarks and 82% for the CFP2006 benchmarks, which was much more than the PC/DC and G/DC prefetchers achieved. The PC/DC prefetcher reduced misses by over 36% and 57% on average for the CINT2006 and CFP2006 benchmarks respectively. The G/DC prefetcher achieved over 35% and 50% on average for these two sets of benchmarks. Cache misses were reduced significantly for most benchmarks with the GMC prefetching, including five benchmarks whose misses were reduced by over 90%. 458.sjeng was the only benchmark on which the GMC prefetcher had little performance improvement (less than 3% miss reduction). This benchmark is very sensitive to cache size and pollution. We conduct a detailed study of cache pollution in the next subsection.

4.3.2 IPC speedup. Figure 15 shows the IPC speedup results when using the GMC prefetcher. The IPC results were reported by the simulator with add-on prefetcher modules that computed prefetch timing and overheads. The bars shown in each group include GMC prefetching, PC/DC prefetching and G/DC prefetching, from left to right. The simulation results clearly show that the GMC prefetcher reduced the average data access latency and improved IPC considerably compared to the other prefetchers. The IPC performance improvement peaked at 259%, and the overall geometric mean of the IPC improvement for the CINT2006 and CFP2006 benchmarks was over 55% and 44% respectively. Compared to the PC/DC and G/DC prefetchers' average IPC speedups (both improved the CINT2006 and CFP2006 benchmarks by around 26% and 25%), the GMC prefetcher was clearly more effective in hiding the memory access latency and improving the application performance.

These results also show that the GMC prefetcher's efficiency is related to the miss rate of the benchmarks. It tends to be more efficient when dealing with benchmarks with a high miss rate. Among all the benchmarks with significant improvement, the average miss rate was 71%. However, the benchmarks on which the GMC prefetcher did not achieve significant miss rate reduction had an average miss rate of only 29%. There was only one exception to this trend, namely 403.gcc with miss rate 75%. Although it had a high miss rate, the prefetcher did not gain much performance on it. Our detailed analysis shows that the 403.gcc benchmark had only 0.2 million accesses over 100 million instructions. Applications with infrequent data access such as 403.gcc can hardly be improved by prefetchers even if their miss rate is high. There were three benchmarks, 445.gobmk, 458.sjeng, and 473.astar, that experienced a slight performance decrease using GMC. These benchmarks exhibited much less predictable access patterns than the others and did not offset the prefetching overhead sufficiently. Since their performance differences were no more than 6%, we consider GMC's overall benefit to outweigh the slight performance disadvantage on this small number of benchmarks.

The simulation results with the SPEC-CPU2006 benchmarks clearly demonstrate the advantage of multi-order and global context analysis for context-based prefetching. They also show that the proposed GMC prefetcher outperformed the recently proposed PC/DC and G/DC prefetchers, with considerable latency reduction for most benchmarks studied.

4.3.3 Sensitivity and cache pollution analysis. Cache pollution is an undesirable side effect of data prefetching techniques. In this subsection we present an analysis of the performance sensitivity and cache pollution of the GMC prefetcher under different cache sizes. To observe the GMC prefetcher's sensitivity to different cache sizes and the impact of potential cache pollution, we considered GMC performance with L2 cache sizes of 512 kB, 1 MB, and 2 MB. Although detailed miss rate reduction results achieved by the GMC prefetcher when simulated with the three different L2 cache sizes are omitted due to space limitations, the results show that a larger cache size helps the GMC prefetcher reduce more misses. A larger cache size lowered the extra misses due to the cache pollution, and therefore reduced the number of misses indirectly. The only exception was the 483.xalancbmk. The miss rate of 483.xalancbmk was no longer high when a larger L2 cache size was used, and thus the GMC prefetcher was less effective, reducing less than 30% of the miss rate. The geometric means of the miss rate reduction rates for the three cache sizes were 55%, 58%, and 60% respectively for the CINT2006 benchmarks and 82%, 83%, and 84% respectively for the CFP2006 benchmarks, showing a slight increase as cache size increased in each case.

Figure 16 shows the IPC speedup results with three different L2 cache sizes. When the L2 cache size was increased, the IPC speedup improved for 437.leslie3d, 473.astar and 482.sphinx3. These benchmarks are memory

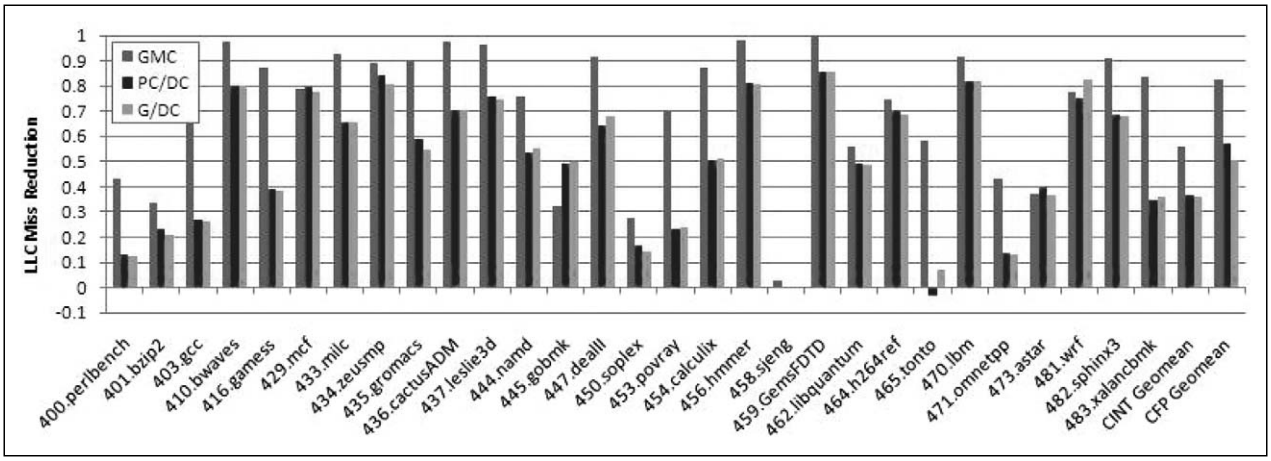

Figure 14. L2 cache miss rate reduction achieved by the GMC, PC/DC and G/DC prefetchers.

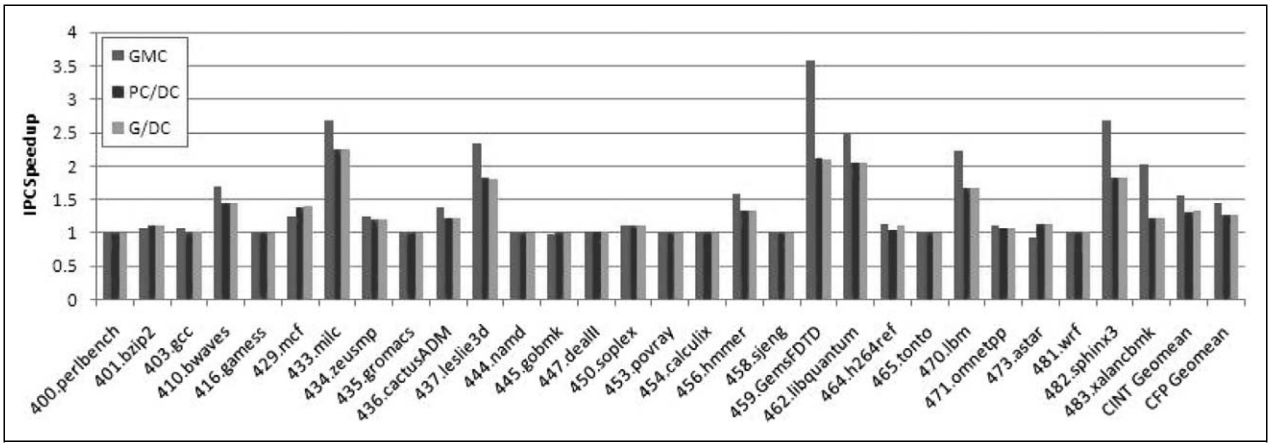

Figure 15. IPC speedup achieved by the GMC, PC/DC and G/DC prefetchers.

access intensive applications and have high miss rates. Large cache sizes hide the performance loss caused by heavy cache pollution for these benchmarks. On the other hand, increasing cache capacity limited the GMC prefetcher's performance when tested with 456.hmmer and 483.xalancbmk because these benchmarks are highly sensitive to cache size. The performance improvement for these benchmarks was due mainly to the enlarged cache capacity rather than the GMC prefetcher. These simulation results and analyses demonstrate that the GMC prefetcher achieved stable performance improvement with different cache sizes and is not highly sensitive to different cache size configurations. Although large cache size helps reduce the cache pollution, the GMC prefetcher's design assigns high priority to highly accurate predictions (high-order analysis and local-context analysis), and can control the cache pollution well.

5 Conclusion

Advances in microprocessor architectures have put more pressure than ever on reducing data access latency for high-end computing systems. Data access latency has been recognized by many as the leading factor preventing high sustained performance of applications, including scientific and high-performance computing applications (Wulf and Mckee, 1995; Vanderwiel and Lilja, 1997; Mckee, 2004; Srinivasan et al., 2004; Nesbit and Smith, 2004; Nesbit et al., 2004; Nesbit and Smith, 2005; Doweck, 2006; Hennessy and Patterson, 2006; Srinath et al., 2007; Diaz and Cintra, 2009, Somogyi et al., 2009). Data prefetching mechanisms, especially the general context-based prefetching approach, have received intensive attention and have been proven to be effective in improving data access performance for high-end computing.

In this study we advance the state-of-the-art in context-based prefetching and further explore its potential. Using extensive simulation testing, we conducted a thorough examination of the performance characteristics of context-based prefetching. We observed that the existing approach with fixed and single order analysis is weak in providing wide prefetching coverage, even though it can potentially achieve high prefetching accuracy, which in turn limits its overall prefetching effectiveness. In addition, the existing approaches do not support both local and

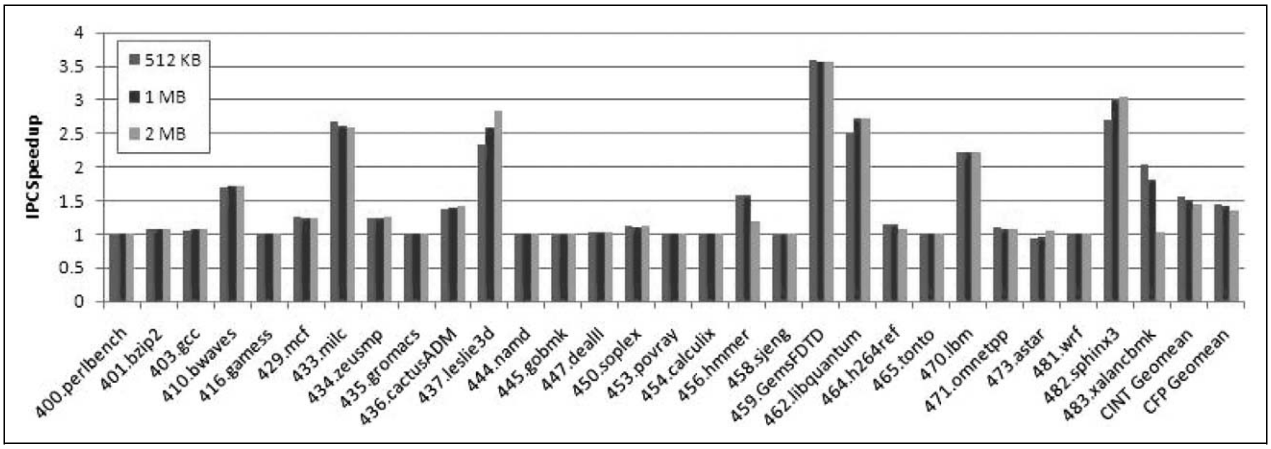

Figure 16. IPC speedup achieved by the GMC prefetcher with 512 kB, 1 MB and 2 MB LLC caches.

global context analysis. Such a drawback limits the overall prefetching effectiveness as well. Motivated by these observations, we proposed to incorporate multi-order, local and global context analysis to improve context-based prefetching, and proposed a new Global-aware and Multi-order Context-based (GMC) prefetcher. We detailed the GMC prefetcher design and the prefetching methodology. We developed prefetch degree and priority control mechanisms to reduce cache pollution and bandwidth contention and to increase the prefetcher's effectiveness. Using simulation experiments with the CMP$im simulator and the SPEC-CPU2006 benchmarks, we confirmed that the GMC prefetcher significantly reduces average data-access latency and increases performance. The GMC prefetcher has the potential to improve data access performance for high-end computing systems.

Funding

This research is sponsored in part by the Office of Advanced Scientific Computing Research, U.S. Department of Energy. This research is also sponsored in part by the National Science Foundation (grant numbers CCF-0621435 and CCF-0937877).

The work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

Conflict of Interest

None declared.

References

Annavaram M, Patel JM, and Davidson ES (2003) Call graph prefetching for database applications. *ACM Transactions on Computer Systems* 21: 412–444.

Bhattacharjee A and Martonosi M (2010) Inter-core cooperative TLB for chip multiprocessors. In: *Proceedings of ASPLOS*, 359–370.

Ceze L, Strauss K, Tuck J, Renau J, and Torrellas J (2006a) CAVA: Using checkpoint-assisted value prediction to hide L2 misses. ACM Transactions on Architecture and Code Optimization.

Ceze L, Tuck J, Torrellas J, and Cascaval C (2006b) Bulk disambiguation of speculative threads in multiprocessors. In: Proceedings of ISCA, 227–238.

Chen S, Kozuch M, Strigkos T, Falsafi B, Gibbons PB, Mowry TC, et al. (2008) Flexible hardware acceleration for instruction-grain program monitoring. In: Proceedings of ISCA, 377–388.

Chen T-F and Baer J-L (1995) Effective hardware based data prefetching for high-performance processors. IEEE Transactions on Computers 44: 609–623.

Chen Y, Byna S, and Sun X-H (2007) Data access history cache and associated data prefetching mechanisms. In: ACM/IEEE Supercomputing Conference.

Chen Y, Zhu H, Jin H, and Sun X-H (2010) Improving the effectiveness of context-based prefetching with multi-order analysis. In: Proceeedings of the 3rd International Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2).

Chou Y (2007) Low-cost epoch-based correlation prefetching for commercial applications. In MICRO.

Dahlgren F, Dubois M, and Stenström P (1993) Fixed and adaptive sequential prefetching in shared memory multiprocessors. In: Proceedings of the ICPP, 56–63.

Dahlgren F, Dubois M, and Stenström P (1995) Sequential hardware prefetching in shared-memory multiprocessors. IEEE Transactions on Parallel and Distributed Systems 6: 733–746.

Diaz P and Cintra M (2009) Stream chaining: exploiting multiple levels of correlation in data prefetching. In: Proceedings of International Symposium on Computer Architecture, 81–92.

Doweck J (2006) Inside Intel Core Micro-architecture and Smart Memory Access. Intel White Paper.

DPC-1 (2008). Data Prefetching Championship Homepage. Available at: http://www.jilp.org/dpc/.

Ebrahimi E, Mutlu O, Lee CJ, and Patt YN (2009) Coordinated control of multiple prefetchers in multi-core systems. In: MICRO, pp. 316–326.

Emma PG, Hartstein A, Puzak TR, and Srinivasan V (2005) Exploring the limits of prefetching. IBM Journal of Research and Development 49: 127–44.

Goeman B, Vandierendonck H, and Bosschere KD (2001) Differential FCM: increasing value prediction accuracy by improving table usage efficiency. In: Proceedings of HPCA.

Hennessy J and Patterson D (2006) Computer Architecture: A Quantitative Approach, 4th ed. San Francisco, CA: Morgan Kaufmann.

Hu Z, Martonosi M, and Kaxiras S (2002) Timekeeping in the memory system: predicting and optimizing memory behavior. In: Proceedings of ISCA, pp. 209–220.

Jaleel A, Cohn RS, Luk C-K, and Jacob B (2008) CMP$im: A pin-based on-the-fly multi-core cache simulator. In: Proceedings of the Fourth Annual Workshop on Modeling, Benchmarking and Simulation.

Joseph D and Grunwald D (1997) Prefetching using Markov predictors. In: Proceedings of International Symposium on Computer Architecture, pp. 252–263.

Jouppi NP (1990) Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In: Proceedings of ISCA, pp. 364–373.

Kandiraju GB and Sivasubramaniam A (2002) Going the distance for TLB prefetching: an application-driven study. In: Proceedings of International Symposium on Computer Architecture.

Lai A-C, Fide C, and Falsafi B (2001) Dead-block prediction and dead-block correlating prefetchers. In: Proceedings of ISCA, 144–54.

Le H, Starke W, Fields J, O'Connell F, Nguyen D, Ronchetti B, et al. (2007) The POWER6 Microarchitecture. IBM Technical White Paper.

Luk C-K, Cohn RS, Muth R, Patil H, Klauser A, Lowney PG, et al. (2005) Pin: building customized program analysis tools with dynamic instrumentation. In: Programming Language Design and Implementation, 190–200.

McKee SA (2004) Reflections on the memory wall. In: Proceedings of the Conference on Computing Frontiers, p. 162.

Nesbit KJ, Dhodapkar AS, and Smith JE (2004) AC/DC: an adaptive data cache prefetcher. In: Parallel Architectures Compilation Techniques, pp. 135–145.

Nesbit KJ and Smith JE (2004) Data cache prefetching using a global history buffer. In: International Symposium on High Performance Computer Architecture, pp. 96–105.

Nesbit KJ and Smith JE (2005) Data cache prefetching using a global history buffer. IEEE Micro 25: 90–97.

Palacharla S and Kessler RE (1994) Evaluating stream buffers as a secondary cache replacement. In: International Symposium on Computer Architecture, pp. 24–33.

Ramos L, Briz J, Ibanez P, and Vinals V (2007) Data prefetching in a cache hierarchy with high bandwidth and capacity. ACM Computer Architecture News, 37–44.

Ramos L, Briz J, Ibanez P, and Vinals V (2009) Multi-level adaptive prefetching based on performance gradient tracking. In: The 1st JILP Data Prefetching Championship.

Sazeides Y and Smith J (1997a) Implementations of Context-based Value Predictors. Technical Report ECE97-8, Department of Electrical and Computer Engineering, University of Wiscosin-Madison.

Sazeides Y and Smith JE (1997b) The predictability of data values. In: International Symposium on Microarchitecture, pp. 248–258.

Somogyi S, Wenisch TF, Ailamaki A, and Falsafi B (2009) Spatio-temporal memory streaming. In: International Symposium on Computer Architecture, pp. 69–80.

Spradling CD (2007) SPEC CPU2006 Benchmark Tools. SIGARCH Computer Architecture News.

Srinath S, Mutlu O, Kim H, and Patt YN (2007) Feedback directed prefetching: improving the performance and bandwidth-efficiency of hardware prefetchers. In: HPCA, pp. 63–74.

Srinivasan V, Davidson ES, and Tyson GS (2004) A prefetch taxonomy. IEEE Transactions on Computers 53: 126–140.

Vanderwiel SP and Lilja DJ (1997) When caches aren't enough: data prefetching techniques. IEEE Computer 30: 23–30.

Wenisch TF, Somogyi S, Hardavellas N, Kim J, Ailamaki A, and Falsafi B (2005) Temporal streaming of shared memory. In: Proceedings of ISCA, pp. 222–233.

Wulf WA and Mckee SA (1995) Hitting the memory wall: implications of the obvious. Co*mputer Architecture News* 23: 20–24.

Author's Biographies

Yong Chen is an Assistant Professor in the Computer Science Department of the Texas Tech University. He received his B.E. degree in Computer Engineering in 2000 and M.S. degree in Computer Science in 2003, both from University of Science and Technology of China, and his Ph.D. degree in Computer Science from Illinois Institute of Technology in 2009. Prior to joining TTU, Dr. Chen worked in the Future Technologies Group of the Computer Science and Mathmetics Division at the DOE Oak Ridge National Laboratory. Dr. Chen's research interests include parallel and distributed computing, high-performance computing, computer architectures and systems software. More information about Dr. Chen can be found at http://www.myweb.ttu.edu/yonchen/.

Huaiyu Zhu received his B.E. degree in Software Engineering in 2007 from Beijing University of Aeronautics and Astronautics and M.S. degree in Computer Science in 2009 from Illinois Institute of Technology. He is currently a Ph.D student at the Electrical and Computer Engineering department of the University of Illinois at Urbana Champaign. His research focuses on high-performance computing, simulation, security and computer architecture in general.

Philip C. Roth is a Computer Scientist at Oak Ridge National Laboratory (ORNL), where he is a founding member of the Future Technologies Group in the Computer Science and Mathematics Division. His research interests include performance analysis, prediction, and tools with special emphases on scalability and automation; emerging technologies for high performance computing; and storage for large-scale systems. He earned his Ph.D. in Computer Science from the University of Wisconsin, Madison in 2005. He joined ORNL in 2004.

Hui Jin received his B.S. degree in Mathematics in 2003 from Wuhan University China, M.S Degree in Computer Science in 2008 from Illinois Institute of Technology. He is currently pursuing his Ph.D. degree in Computer Science from Illinois Institute of Technology. His research focuses on parallel and distributed computing and computer architecture in general, and the reliability of HPC systems, including performance evaluation and optimization. More information about him can be found at http://www.iit.edu/~hjin6.

Xian-He Sun is a Professor of Computer Science and the Chairman of the Department of Computer Science at the Illinois Institute of Technology (IIT). He is the director of the Scalable Computing Software laboratory at IIT, and is a guest faculty in the Mathematics and Computer Science Division and Computing Division at the Argonne and Fermi National Laboratory, respectively. Before joining IIT, he worked at DoE Ames National Laboratory, at ICASE, NASA Langley Research Center, and at Louisiana State University, Baton Rouge. Dr. Sun's research interests include parallel and distributed processing, software systems, performance evaluation, and data intensive computing. More information about Dr. Sun can be found at http://www.cs.iit.edu/~sun/.