Coterminous Locality and Coterminous Group Data Prefetching on Chip-Multiprocessors

Xudong Shi[1], Zhen Yang[1], Jih-Kwon Peir[1], Lu Peng[2], Yen-Kuang Chen[3], Victor Lee[3], Bob Liang[3]

[1]Computer & Information Science & Engineering
University of Florida
Gainesville, FL 32611, USA
{xushi, zhyang, peir}@cise.ufl.edu

[2]Electrical & Computer Engineering
Louisiana State University
Baton Rouge, LA 70803, USA
lpeng@lsu.edu

[3]Architecture Research Lab
Intel Corporation
Santa Clara, CA 95052, USA
{yen-kuang.chen, victor.w.lee, bob.liang}@intel.com

Abstract

Due to shared cache contentions and interconnect delays, data prefetching is more critical in alleviating penalties from increasing memory latencies and demands on Chip-Multiprocessors (CMPs). Through deep analysis of SPEC2000 applications, we find that a part of the nearby data memory references often exhibit highly-repeated patterns with long, but equal block reuse distance. These references can form a coterminous group (CG). Coterminous locality is introduced as that when a member in a CG is referenced, the remaining members will likely be referenced in the near future. Based on the coterminous locality behavior, we implement a novel CG data prefetcher on CMPs. Performance evaluations show that the proposed prefetcher can accurately cover up to 40-50% of the total misses, and result in 50-60% of potential performance improvement for several selected workload mixes.

1. Introduction

As VLSI circuit integration continues to advance with deep submicron technology, billions of transistors will be available in a single processor die with a clock frequency exceeding 10 GHz. Because of limited Instruction-Level Parallelism (ILP), design complexities, as well as high energy/power consumptions, further expanding wide-issued, out-of-order single-core processors with huge instruction windows and super-speculative execution techniques will suffer diminishing returns. It becomes a norm that processor dies will contain multiple cores with shared caches for a higher chip-level Instruction-Per-Cycle (IPC) [25,27]. In a Chip-Multiprocessor (CMP), however, contentions of shared resources significantly hamper performance of individual threads and hinder exploiting parallelism from multiple threads [14].

Speculative data prefetching techniques become more critical on CMPs for hiding the longer memory latency problem from heavier cache contentions and limited memory bandwidth. Traditional data prefetching based on cache miss correlations [6,11] faces serious obstacles. First, each cache miss often has several potential successive misses and prefetching multiple successors is inaccurate and expensive. Such incorrect speculations are more harmful on CMPs, wasting memory bandwidth and polluting critical shared caches. Second, consecutive cache misses can be separated by few instructions. It could be too late to initiate prefetches for successive misses. Third, reasonable miss coverage requires long history which translates to more power/area.

For the purpose of accuracy and timeliness, stream-based approaches dynamically identify repeated streams of cache misses [8,9,31]. The hot-stream prefetcher [8,9] profiles and analyzes sampled memory traces on-line to identify frequently repeated sequence (hot streams). Hot streams are prefetched by prefetching instructions inserted into the binary. However, periodic profiling, analysis, and binary code insertions incur execution overheads, which may become excessive for streams with long reuse distances. The temporal-stream prefetcher [31] identifies recent streams that start with the current miss and prefetches the most probable stream with repeated miss sequences. However, a large FIFO buffer is required to record the miss history for identifying streams.

1-4244-0054-6/06/$20.00 ©2006 IEEE

In this paper, we propose a *Coterminous Group (CG)* based data prefetching technique on CMPs to improve the overall system performance. Our analysis of SPEC applications shows that adjacent traversals of various data structures, such as arrays, trees and graphs, often exhibit *temporal* repeated memory access patterns. A unique feature of these nearby accesses is that they exhibit a long but equal reuse distance. We define such a group of memory references as a *Coterminous Group (CG)* and *the highly repeated access patterns among members in a CG as coterminous locality.* The CG-prefetcher identifies and records highly repeated CGs in a small buffer for accurate and timely prefetches for members in a group.

The paper makes three main contributions. First, we demonstrate the severe cache contention problem with various mixes of SPEC2000 applications, and describe the necessities and the challenges of data prefetching on CMPs. Second, we discover the existence of coterminous groups in these applications and quantify the highly repeated coterminous locality among members in a CG. Third, based on coterminous group, we develop a new prefetching scheme, *CG-prefetcher,* and present a realistic implementation by integrating the CG-prefetcher into the memory controller. Full system evaluations have shown that the proposed CG-prefetcher can accurately prefetch the needed data in a timely manner on CMPs. It generates about 10-40% extra traffic to achieve 20-50% of miss coverage in comparison with 2.5 times more extra traffic by a correlation-based prefetcher with a comparable miss coverage. The CG-prefetcher also shows better IPC improvement than the correlation-based or the stream-based prefetchers.

The remainder of this paper is organized as follows. Section 2 describes the severe cache contention problems on CMPs. Section 3 shows the coterminous group and coterminous locality. Section 4 develops the basic design of the memory-side CG-prefetcher. Section 5 presents the simulation methodology. This is followed by performance evaluations in Section 6. Related work is given in Section 7 followed by a brief conclusion in Section 8.

2. Cache Contentions on CMPs

Figure 1 shows the IPCs of a set of SPEC2000 workloads that are running independently, or in parallel on 2- or 4-core CMPs. The first three groups are 4-workload mixes of *Art/Mcf/Ammp/Twolf*, *Art/Mcf /Vortex/Bzip2*, and *Twolf/Parser/Vortex/Bzip2*. The first group consists of workloads with heavier L2 misses; the second group mixes workloads with heavier and lighter L2 penalties; and the third group has workloads with lighter L2 misses. For each group, the workloads are ordered by high-to-low L2 miss penalties from left to right in their appearance. We also run nine 2-workload mixes, also ranging from high-to-low L2 miss penalties, including *Art/Mcf, Mcf/Mcf, Mcf/Ammp, Art/Twolf, Mcf/Twolf, Mcf/Bzip2, Twolf/Bzip2, Parser/Bzip2,* and *Bzip2/Bzip2*. Detailed descriptions of the simulation model and workload selection will be given in Section 5.

Significant IPC reductions can be observed on individual workloads when they run in parallel, especially for the workload mixes with high contentions on shared caches. For example, when *Art/Mcf/Ammp/Twolf* are running on four cores, the individual IPCs drop from 0.029, 0.050, 0.132, and 0.481 to 0.022, 0.026, 0.043, and 0.181 respectively. Instead of accumulating the overall IPCs on four cores, the IPC is dropped to only 40% from 0.69 to 0.27. Similar effects of various degrees can also be observed with two cores. These significant IPC degradations demand for accurate prefetchers to alleviate heavier cache contentions and misses on CMPs.

3. Coterminous Group and Locality

A *Coterminous Group (CG)* consists of nearby data references with *same block reuse distances*. The *block reuse distance* is defined as the number of distinct data blocks that are referenced between two consecutive references to the same block. For instance, consider the following accessing sequence: ***a b c*** *x* ***d*** *x y z* ***a b c*** *y* ***d***. The reuse distances of a-a, b-b, c-c and d-d are all 6, whereas x-x is 1 and y-y is 4. In this case, ***a b c d*** can form a CG. References in a CG have three important properties. First, the order of references must be exactly the same at each repetition (e.g. *d* must follow *c*, *c* follows *b* and *b* follows *a*). Second, references in a CG can interleave with other references (e.g. *x, y*). These references, however, are difficult to predict accurately, and will be excluded by the criteria of same distance. Third, the same reference (i.e. to the same block) usually does not appear twice in one CG.

Figure 2 plots reuse distances of 3000 nearby references from three SPEC2000 applications. The existence of CGs is quite obvious from these snapshots. *Mcf* has a huge CG with a reuse distance of over 60,000. *Ammp* shows four large CGs along with a few small ones. And *Parser* has many small CGs. Note that references with short reuse distances (e.g. < 512), which are usually covered by temporal and spatial localities, are filtered. Other applications also show the CG behavior. We only present three examples due to the space limit.

Based on these behaviors, we conclude that members in a CG exhibit a highly repeated access pattern, i.e. *whenever a member in a CG is referenced, the remaining members will likely be referenced in the near future according to the previous accessing sequence.* We call such highly repeated patterns *coterminous locality.*

We can quantify the coterminous locality by measuring the *pair-wise correlation A->B between adjacent references* in a CG. (This is similar to the miss correlation

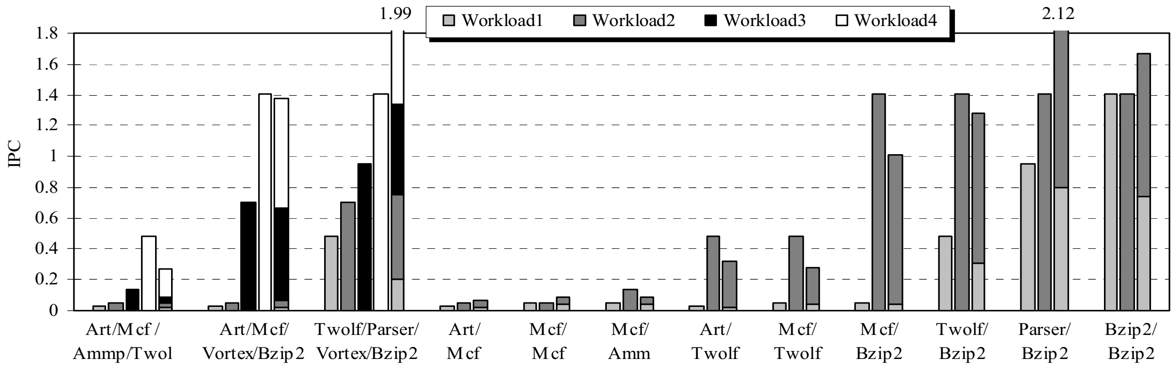

Figure 1. IPCs of workload mixes on CMPs

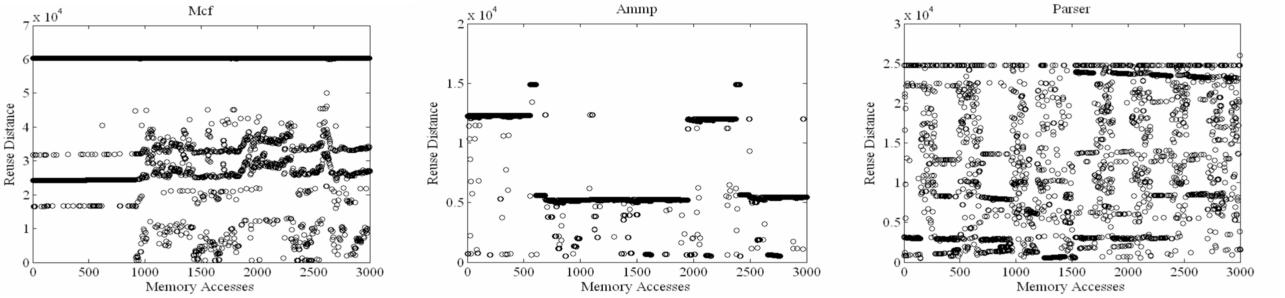

Figure 2. Reuse distances of three SPEC 2000 applications

in [6,11].) In measuring the locality, up to four successors of each reference are kept based on the LRU replacement. The accuracy of *B* being referenced immediately after the re-reference of *A* provides a good locality measurement. When a match to the successor in the MRU position indicates a repeated reference of *A* followed by *B*.

We also relax the reuse distance requirement. *CG-N* represents CGs, in which nearby references with reuse distances that are within ±N. CG-0 represents the original same-distance CG, while CG-∞ has a single CG that includes all references with long block reuse distances.

Figure 3 shows the accumulated percentages of repeats to the four successors, with two different scales of Y-axis to improve readability. In general, the results exhibit strong repeated reference behaviors among members in a CG. Due to array accesses, *Ammp* shows nearly perfect correlations regardless of the reuse distance requirement. *Art* exhibits high correlations, especially for CG-0. All others also demonstrate strong correlations. As expected, CG-0 shows stronger correlations than other weaker forms of CGs, while CG-∞, which is essentially the same as the adjacent cache-miss correlation, shows very poor correlations. The gap between CG-0 and CG-2/CG-4/CG-8 is rather narrow in *Mcf, Vortex,* and *Bzip2*, suggesting a weaker form of CGs may be preferable for covering more references. A large gap is observed between CG-0 and other CGs in *Twolf*, *Parser*, and *Gcc* indicating CG-0 is more accurate for prefetching.

4. Memory-side CG-prefetcher on CMPs

Based on existences of highly-repeated coterminous locality within members in CGs, we design and integrate a CG-prefetcher in CMP memory controllers. Although it is suitable on uni-processor systems too, the accurate CG-prefetcher is more appealing on emerging CMPs due to extra resource contentions and constraints.

4.1. Basic Design of CG-Prefetcher

The structure of a CG-prefetcher is illustrated in Figure 4. A *Request FIFO* records the block addresses and their reuse distances of recent memory requests. A CG starts to form once the number of requests with the same reuse distance in the *Request FIFO* exceeds a certain threshold. The threshold controls the aggressiveness of forming a new CG, and the size of the FIFO determines the adjacency of members. A flag is associated with each request indicating whether the request is matched. The matched requests in the FIFO are copied into a *CG Buffer* waiting for the CG to be formed. The size of the CG buffer determines the maximum number of members in a CG, which can control the timeliness of prefetches. A small number of CG Buffers allows multiple CGs to be formed concurrently. A CG is completed when either the CG Buffer is full or a new CG is identified from the *Request FIFO*. In either case, the old CG is moved to the

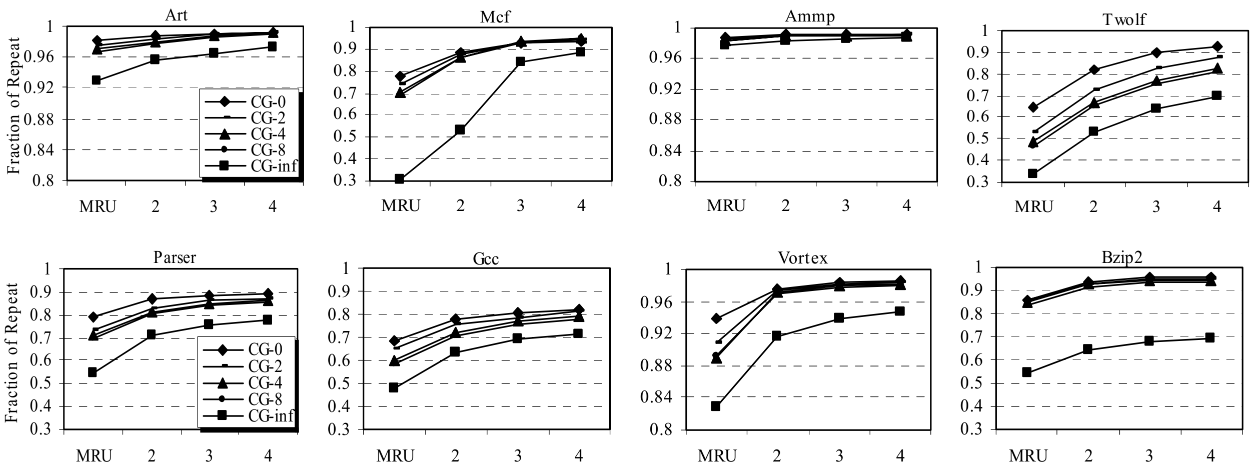

Figure 3. Correlations of adjacent references within CGs

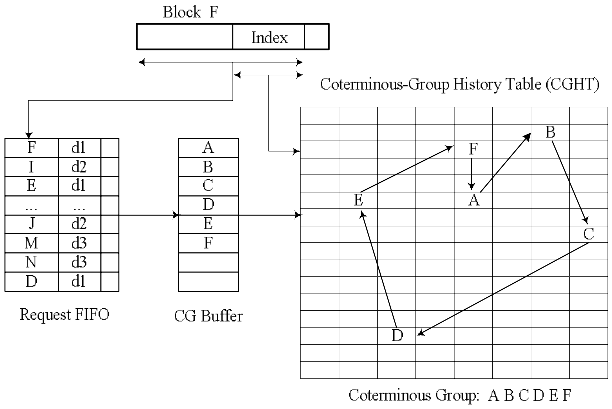

Figure 4. Structure of CG prefetcher

Coterminous Group History Table (CGHT), a set-associative directory indexed by block addresses. A unidirectional pointer in each entry links the members in a CG. This link-based CGHT permits fast searching of a CG from any member in the group. When the CGHT becomes full, either the LRU entries are replaced and removed from the existing CGs, or the conflicting new CG members are dropped to avoid potential thrashing.

Once a L2 miss hits the CGHT, the entire CG can be identified and prefetched by following the circular links. In Figure 4, for instance, a miss to block F will trigger prefetches of A, B, C, D, and E in order. Note that a block can appear in more than one CG in the CGHT. This is because a block reference can leave a CG and become a member of a new CG, while the CGHT may still keep the old CG. On multiple hits, either the most recent CG or all the matched CGs may be prefetched. Any existing CGs can change dynamically over a period of time. Updating CGs dynamically is difficult without precise information on when a member leaves or joins a group. Another option is to simply flush the CGHT periodically based on the number of executed instructions, memory references, or cache misses. However, a penalty will be paid to reestablish the CGs after each flush.

4.2. Integrating CG-prefetcher on CMP Memory Systems

There are attractive features of a memory-side CG-prefetcher [26]. First, it minimizes changes to the complex processor pipeline along with any associated performance and space overheads. Second, it may use the DRAM array to store necessary state information with minimum cost. A recent trend is to integrate the memory controller in the processor die to reduce interconnect latency. Nevertheless, such integration has minimal performance implication on implementing the CG-prefetcher in the memory controller.

The first key issue for a memory-side CG-prefetcher is to determine the block reuse distance without seeing all processor requests at the memory controller. A *global miss sequence number* is used. The memory controller assigns and saves a new sequence number to each missed memory block in the DRAM array. The reuse distance can be approximated as the difference of the new and the old sequence numbers. For a 128-byte block with a 16-bit sequence number, a reuse distance of 64K blocks, or an 8MB working set can be covered. The memory overhead is merely 1.5%. When the same distance requirement is relaxed, one sequence number can be for a small number of adjacent requests, which will expand the working set coverage and/or reduce the space requirement.

Figure 5 shows the CG-prefetcher in memory system. To avoid regular cache-miss requests from different cores disrupting one another for establishing the CGs [28], we construct a private CG-prefetcher for each core. Each CG-prefetcher has a *Prefetch Queue (PQ)* to buffer the prefetch requests (addresses) from the associated prefetcher. A shared *Miss Queue (MQ)* stores regular miss requests from all cores for accessing the DRAM channels. A shared *Miss Return Queue (MRQ)* and a shared *Prefetch Return Queue (PRQ)* buffers the data from the miss requests and the prefetch requests for accessing the memory bus. We implement a private PQ to prevent prefetch requests of one core from blocking those from other cores. The PQs have lower priority than the MQ. Among the PQs, a round-robin fashion is used. Similarly, the PRQ has lower priority than the MRQ in arbitrating the system bus. Each CG-prefetcher maintains a separate sequence number for calculating the block reuse distance.

When a regular miss request arrives, all the PQs are searched. In case of a match, the request is removed from the PQ and is inserted into the MQ, gaining a higher priority to access the DRAM. In this case, there is no performance benefit since the prefetch of the requested block has not been initiated. If a matched prefetch request is in the middle of fetching the block from the DRAM, or is ready in the PRQ, waiting for the shared data bus, the request will be redirected to the MRQ for a higher priority to arbitrate the data bus. Variable delay cycles can be saved depending on the stage of the prefetch request. The miss request is inserted into the MQ normally when no match is found.

A miss request can trigger a sequence of prefetches if it hits the CGHT. The prefetch requests are inserted into the corresponding PQ. If the PQ or the PRQ is full, or if a prefetch request has been initiated, the prefetch request is simply dropped. In order to filter the prefetched blocks already located in processor's cache, a topologically equivalent directory of the lowest level cache is maintained in the controller (not shown in Figure 5). The directory is updated based on misses, prefetches, and write-backs to keep it close to the cache directory. A prefetch is dropped in case of a match. Note that all other simulated prefetchers incorporate the directory too.

5. Evaluation Methodology

5.1. Simulators and Parameters

We use Virtutech Simics 2.0, a full-system execution-driven simulator, to model an out-of-order Pentium 4 Linux machine. Simics is configured to support chip multiprocessors, with each core having its own L1 cache and all cores sharing a unified L2 cache. We add a *g-share* branch predictor and an independent stride prefetcher to each core.

We implement a cycle-by-cycle event-driven memory simulator to accurately model the memory system. Multi-channel DDR SDRAM is simulated. The DRAM accesses are pipelined whenever possible. A cycle-accurate, split-transaction processor-memory bus is also included. All timing delays of misses and prefetches are carefully simulated. Due to a slower clock of the memory controller, the memory-side prefetchers initiate one prefetch every 10 processor cycles. Table 1 summarizes the important simulation parameters.

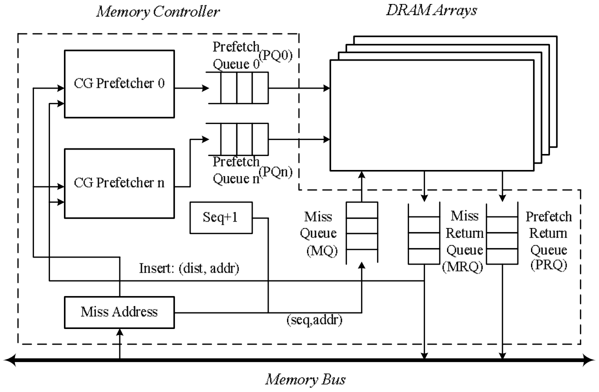

Figure 5. Basic design of a memory-side CG

5.2. Workload Selection

We use several mixtures of SPEC2000 benchmark workloads based on the classification of memory-bound and CPU-bound workloads [32]. The memory-bound workloads are *Art, Mcf,* and *Ammp*, while the CPU-bound workloads are *Twolf, Parser, Vortex,* and *Bzip2*. The first category of workload mixes, MEM, includes memory-bound workloads; the second category of workload mixes, MIX, consists of both memory-bound and CPU-bound workloads; and the third category of workloads, CPU, contains only CPU-bound workloads. We choose the *ref* input set for all the SPEC2000 workloads. Table 2 summarizes the selected workload mixes.

We skip certain instructions for each individual application in a mix based on studies done in [22], and run the workload mix for another 100 million instructions for warming up the caches. A Simics checkpoint for each mix is generated afterwards. We run our base simulator, without any memory-side prefetcher, until any application in a mix has executed at least 100 million instructions for collecting instruction distributions [24]. Such instruction distributions are then applied to all prefetchers to collect statistics.

5.3. Prefetcher Configurations

The performance results of the proposed CG-prefetcher are presented and compared against a pair-wise miss-correlation prefetcher (*MC-prefetcher*), a prefetcher based on the last miss stream (*LS-prefetcher*), and a hot-stream prefetcher (HS-prefetcher). A processor-side stride prefetcher is included in all simulated prefetchers.

Processor-side Stride Prefetcher: It has 4k-entry PCs with each entry maintaining four previous references of

CMP: 2 or 4 cores, 3.2GHz
ROB size: 128
Fetch/Exec/Retire/Commit width: 4 / 7 / 5 / 3
Branch predictor: G-share, 64KB, 4K BTB
Branch misprediction penalty: 10 cycles
Processor side prefetcher: Stride
L1-I: 64KB, 4-way, 64B Line, MESI
L1-D: 64KB, 4-way, 64B Line, MESI
L2: 1MB, 8-way, 64B Line
L1-I/L1-D/L2 latency: 0/2/15 cycles
L1/L2 MSHR size: 16/16
Memory latency: 432 cycles
DRAM channels: 2/4/8/16
Queue size (MQ, PQ_n, PRQ, MRQ) : 16
Memory side prefetcher: None/CG/MC/HS/LS
DRAM access latency: 180 cycles
Interconnection latency: 220 cycles
History table search latency: 10 cycles/entry
Memory bus: 8-byte, 800MHz, 6.4GB/s

Table 1. Simulation parameters

	MEM	MIX	CPU
Four	Art/Mcf/ Ammp/Twolf	Art/Mcf/ Vortex/Bzip2	Twolf/Parser/ Vortex/Bzip2
Two	Art/Mcf	Art/Twolf	Twolf/Bzip2
	Mcf/Mcf	Mcf/Twolf	Parser/Bzip2
	Mcf/Ammp	Mcf/Bzip2	Bzip2/Bzip2

Table 2. Selected workload mixes

that PC. Four successive prefetches are issued, whenever four stride distances of a specific PC are matched [15].

Memory-side MC-prefetcher: Each core has a MC-prefetcher with a 128k-entry 8 set-associative history table. Each miss address (each entry) records 2 successive misses. Upon a miss, the MC-prefetcher prefetches two levels in depth, resulting in a total of up to 6 prefetches.

Memory-side HS-prefetcher: The HS-prefetcher is simulated based on a Global History Buffer [20,31] with 128k-entry FIFO and 64k-entry 16 set-associative miss index table for each core. Each FIFO entry consists of a 26-bit block address and a 17-bit pointer that sequentially links the entries with the same miss address. On every miss, the index and the FIFO are searched sequentially to find all recent streams that begin with the current miss. If the first 3 misses of any two streams match, the matched stream is prefetched. The length of each stream is 8.

Memory-side LS-prefetcher: The LS-prefetcher is a special case of the HS-prefetcher, where the last miss stream is prefetched without any further qualification.

Memory-side CG-Prefetcher: We use CG-2 to get both high accuracy and decent coverage of misses. The CGHT is 16k entries per core, with 30 bits (16-way set-associative) per entry. We use a 16-entry Request FIFO and four 8-entry CG-Buffers. A CG can be formed once three memory requests in the Request FIFO satisfy the reuse distance requirement. Each CG contains up to 8 members. The CGHT is flushed periodically every 2 million misses from the corresponding core.

	Memory Controller (SRAM) per core	DRAM
CG	60KB(16K*30bit/8)	3%
MC	2MB(128K*2*64bit/8)	0
HS	920KB(128K*43bit/8+64K*29bit/8)	0
LS	920KB(128K*43bit/8+64K*29bit/8)	0

Table 3. Space overhead for various prefetchers

Table 3 summarizes the extra space overhead to implement various prefetchers.

6. Performance Evaluation

6.1. IPC Improvement and Miss Reductions

In Figure 6, the IPC speedups of Stride-only, MC, HS, LS, and CG prefetchers with respect to the baseline model are presented. (IPCs for the baseline model were given in Figure 1.) Each IPC-speedup bar is broken into the contributions made by individual workloads in the mix. The total height represents the overall IPC speedup.

Several observations can be made. First, most workload mixes show performance improvement for all five prefetching techniques. In general, the CG has the highest overall improvement, followed by the LS, the HS, and the MC prefetchers. Two workload mixes *Art/Twolf* and *Mcf/Twolf* show a performance loss for most prefetchers. Our studies indicate that *Twolf* has irregular patterns, and hardly benefits from any of the prefetching schemes. Although *Art* and *Mcf* are well performed, the higher IPC of *Twolf* dominates the overall IPC speedup. Second, the CG-prefetcher is a big winner for the MEM workloads with speedup of 40% in average, followed by the LS with 30%, the HS with 24% and the MC with 18%. The MEM workloads exhibit heavier cache contentions and misses. Therefore, the accurate CG-prefetcher benefits the most for this category. Third, the CG-prefetcher generally performs better in the MIX and the CPU categories. However, the LS-prefetcher slightly outperforms the CG-prefetcher in a few cases. With lighter memory demands in these workload mixes, the LS-prefetcher can deliver more prefetched blocks with a smaller impact on cache pollutions and memory traffic.

It is important to note that the measured IPC speedup creates an unfair view when comparing mixed workloads on multi-cores. For example, in *Art/Mcf/Vortex/Bzip2*, the IPC speedups of individual workloads are measured at 3.16, 1.41, 0.82, and 1.42 for the CG-prefetcher, and 2.39, 1.22, 0.86, and 1.49 for the MC-prefetcher. Therefore, the

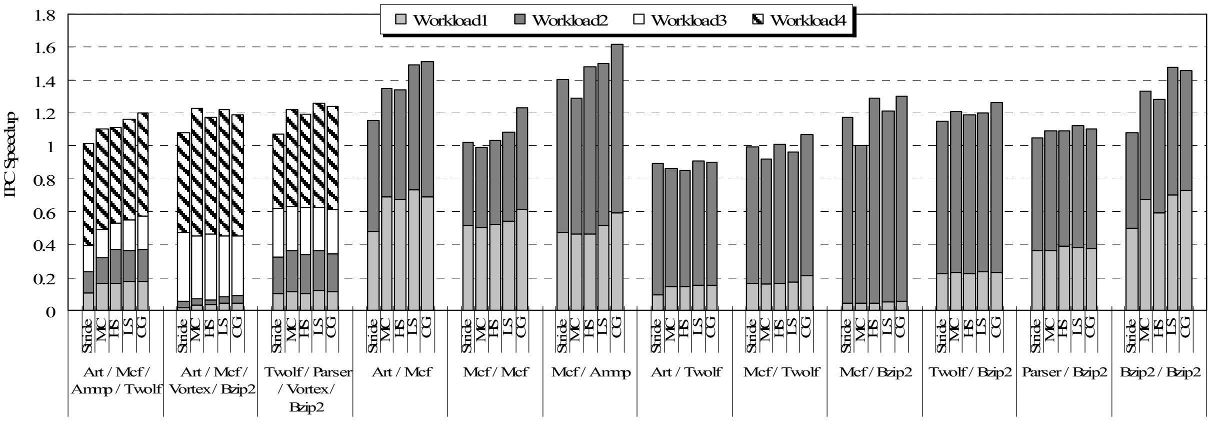

Figure 6. IPC speedups of Stride, MC, HS, LS, CG prefetchers (Normalized to baseline IPC)

average speedups of the four workloads are 1.70 and 1.49 for the two prefetchers. However, their measured IPC speedups are only 1.20 and 1.22. Given the fact that *Vortex* and *Bzip2* have considerably higher IPC than those of *Art* and *Mcf*, the overall IPC improvement is dominated by the last two workloads. This is true for other workload mixes. In Figure 7, the average speedup of two MEM and two MIX workload mixes are shown. Comparing with the measured speedups, significantly higher average speedups are achieved by all prefetchers. For *Art/Twolf*, the average IPC speedups are 48%, 44%, 52% and 51% for the respective MC, HS, LS and CG prefetchers, instead of -14%, -15%, -9%, and -10% as shown in Figure 6.

In contrast to the MC- and the LS-prefetcher, the HS- and the CG-prefetcher carefully qualify members in a group that show highly repeated patterns for prefetching. The benefit of this accuracy is evident in Figure 8. The total memory accesses are classified into 5 categories for each prefetcher: misses, partial hits, miss reductions (i.e. successful prefetches), extra prefetches, and wasted prefetches. The sum of the misses, partial hits, and miss reductions is equal to the baseline misses without prefetching, which is normalized to 1 in the figure. The partial hits mean latency reductions to misses due to earlier but incomplete prefetches. The extra prefetches represent the prefetched blocks that are replaced before any usage. The wasted prefetches refer to the prefetched blocks that are presented in cache already. Overall, all prefetchers show a significant reduction of cache misses ranging from a few percent to as high as 50%.

The MC- and the LS-prefetcher generate significantly higher memory traffic than the HS- and the CG-prefetcher. On the average, the HS-, the CG-, the LS- and the MC-prefetcher produce about 4%, 21%, 35%, and 52% extra traffic respectively. The excessive memory traffic by the LS- and the MC-prefetcher does not turn proportionally into a positive reduction of the cache miss. In some cases, the impact is negative mainly due to the cache pollution problem on CMPs. Between the two, the LS-prefetcher is more effective than the MC-prefetcher indicating prefetching multiple successor misses may not be a good idea. The HS-prefetcher has the highest accuracy. However, the low miss coverage limits its overall IPC improvement.

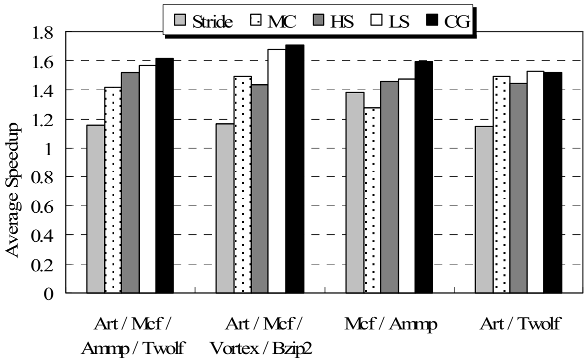

Figure 7. Average speedup of 4 workload mixes

6.2. Sensitivity Studies

The distance constraint of forming a CG is simulated and the performance results of CG-0, CG-2 and CG-8 are plotted in Figure 9. With respect to the measured IPC speedups, the results are mixed. We selected CG-2 to represent the CG-prefetcher due to its slightly better IPCs than that of CG-0 with considerably less traffic than that of CG-8. Note that we omit CG-4, which has similar IPC speedup in comparison with CG-2, but generates more memory traffic.

The impact of group size is evaluated as shown in Figure 10. Two workload mixes in the MEM category, *Art/Mcf/Ammp/Twolf* and *Mcf/Ammp*, and two in the MIX category, *Art/Mcf/Vortex/Bzip2* and *Art/Twolf* are chosen due to their high memory demand. The measured IPCs decrease slightly or remain unchanged for the two 4-workload mixes, while they increase slightly with the two

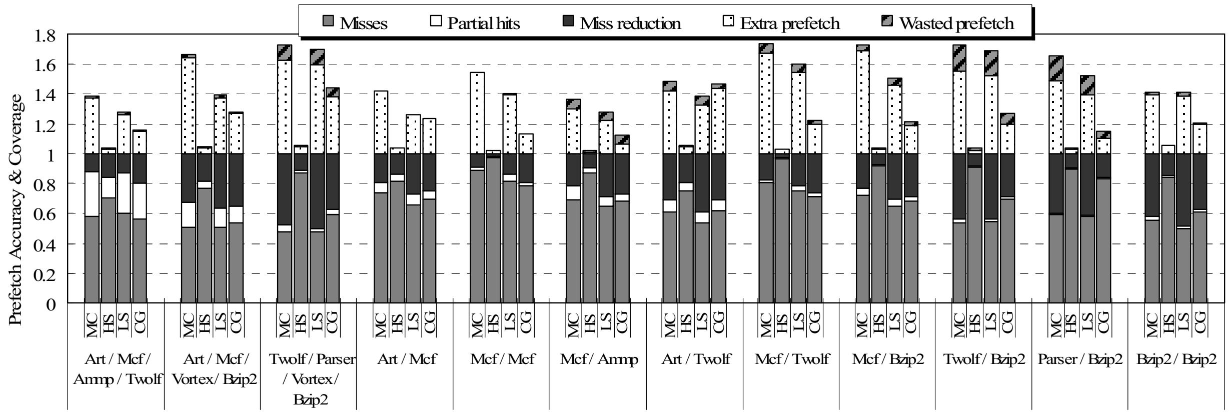

Figure 8. Prefetch accuracy and coverage of simulated prefetchers

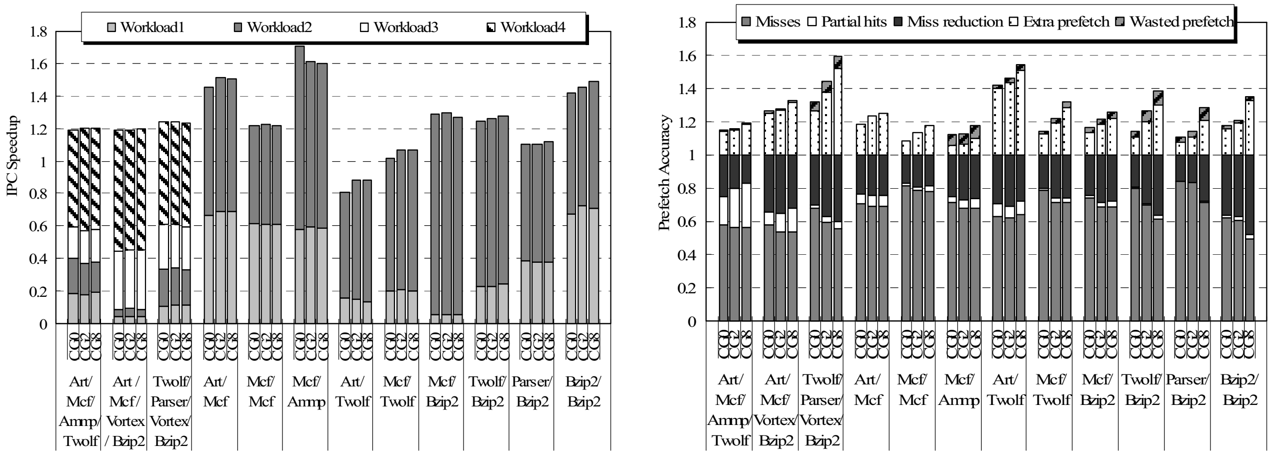

Figure 9. Effect of distance constrains: (Left) Measured IPC speedup; (Right) Accuracy and traffic

2-workload mixes. Due to cache contentions, larger groups generate more useless prefetches. The group size of 8 shows a balance of high IPCs with low overall memory traffic.

Figure 11 plots the average speedup of CG with respect to Stride-only for different L2 cache sizes from 512KB to 4MB. As observed, the four workload mixes behave very differently with respect to different L2 sizes. For *Art/Mcf/Vortex/Bzip2* and *Art/Twolf*, the average IPC speedups are peak at 1MB and 2MB respectively, and then drop sharply afterwards because of a sharp reduction of cache misses with larger caches. However, for the memory-bound workload mixes, *Art/Mcf/Ammp/Twolf* and *Mcf/Ammp*, the average speedups of median-size L2 are slightly less than those of smaller and larger L2. With smaller caches, the cache contention problem is so severe that a small percentage of successful prefetches can lead to significant IPC speedups. For median size caches, the impact of delaying normal miss due to conflicts with prefetches begins to compensate the benefit of prefetching. When the L2 size continues to increase, the number of misses decreases and it diminishes the effect of accessing conflicts. As a result, the average speedup increases again.

Given a higher demand for accessing the DRAM for the prefetching methods, we perform a sensitivity study on the DRAM channels as shown in Figure 12. The results indicate that the number of DRAM channels does show impacts on the IPCs and more so to the memory-bound workload mixes. All four workload mixes perform poorly with 2 channels. However, the improvements are saturated about 4 to 8 channels.

7. Related Work

The single-chip multiprocessor was first presented in [21]. Since then, many companies have announced their multi-core products [2,13,18,1,19]. Trends, opportunities, and challenges for future chip multiprocessors have appeared in keynote speeches, as well as in special columns of recent conferences and professional journals [3,25,27,4], which have inspired the studies in this paper.

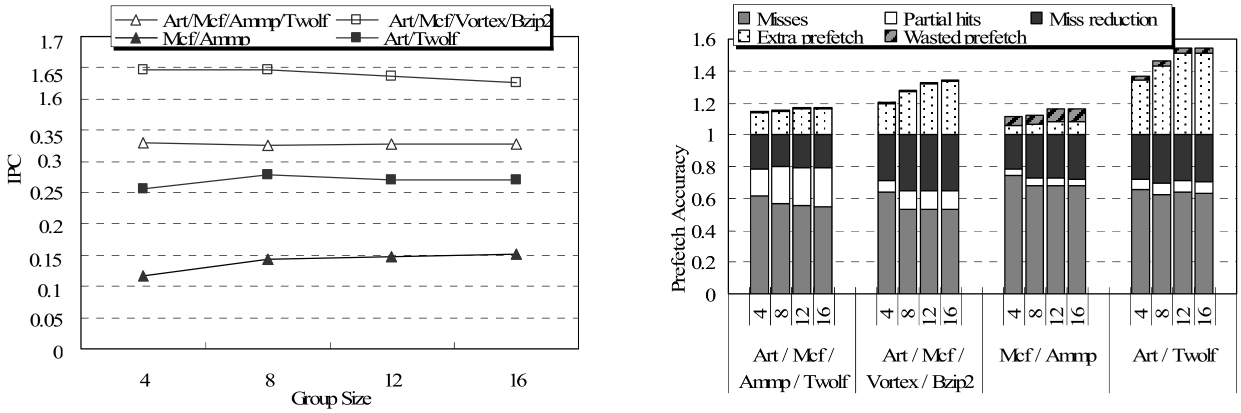

Figure 10. Effect of group size: (left) Measured IPCs; (right) Accuracy and traffic

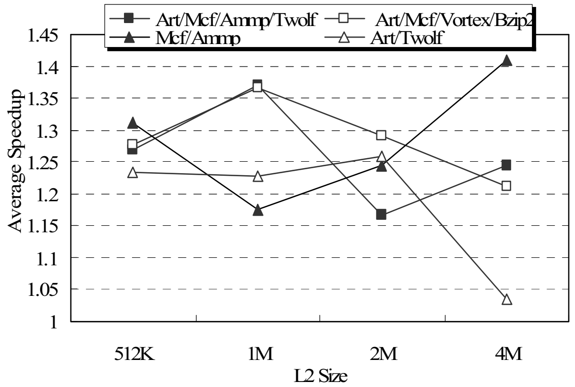

Figure 11. Effect of L2 size

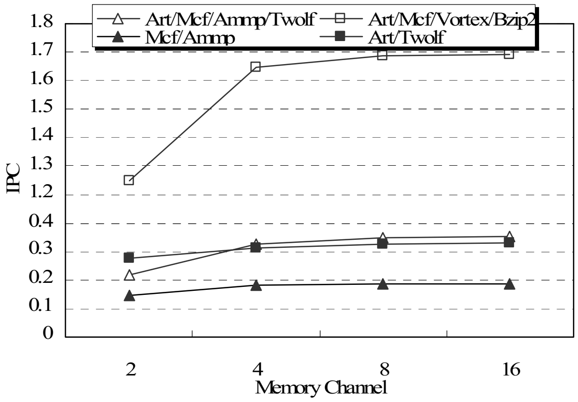

Figure 12. Effect of memory channels

Many uni-processor data prefetching schemes have been proposed in the last decade [29,6,20,30]. Traditional sequential or stride prefetchers work well for workloads with regular spatial access patterns [7,12,15]. Prior correlation-based predictors (e.g. Markov predictor [11, 6] and Global History Buffer [20]) record and use past miss correlations to predict future cache misses. However, a huge history table or FIFO is usually needed to provide decent coverage. Hu et al. [10] uses tag-correlation, a much bigger block correlation, to reduce the history size. To avoid cache pollution and provide timely prefetches, the dead-block prefetcher issues a prefetch once a cache block is predicted to be dead [16].

Chilimbi [8,9] introduced a hot-stream prefetcher. It profiles and analyzes sampled memory traces on-line to identify frequently repeated sequences (hot streams) and inserts prefetching instructions to the binary code for these streams. The profiling, analysis, and binary code insertions / modifications incur execution overheads, and may become excessive to cover hot streams with long reuse distances. Wenisch et al. [31] proposed temporal streams by extending hot streams and global history buffer to deal with coherence misses on SMPs. It requires a huge FIFO and multiple searches/comparisons on every miss to capture repeated streams. The proposed CG prefetcher uses approximated reuse distances to capture repeated coterminous groups with minimum overhead.

Saulsbury et al. [23] proposed a recency-based TLB preloading. It maintains the TLB information in a Mattson stack, and preloads adjacent entries in the stack upon a TLB miss. The recency-based technique can be applied for data prefetching. Compared with the CG-prefetcher, it prefetches adjacent entries in the stack without the prior knowledge of whether the adjacent requests have showed any repeated patterns or how the two requests arrive at the adjacent stack positions. In contrast, the CG approach carefully qualifies members in a group based on the same reuse-distance (i.e. have shown repeated patterns) and physical adjacency (i.e. within a short window) of the requests to achieve higher accuracy.

8. Conclusion

This paper has introduced an accurate CG-based data prefetching scheme on Chip Multiprocessors (CMPs). We showed the existence of coterminous groups (CGs) and a third kind of locality, *coterminous locality*. In particular, the order of nearby references in a CG follows exactly the same order that these references appeared last time, even though they may be irregular. The proposed prefetcher uses CG history to trigger prefetches when a member in a group is re-referenced. It overcomes challenges of the existing correlation-based or stream-based prefetchers, including low prefetch accuracy, lack of timeliness, and large history. The accurate CG-prefetcher is especially appealing for CMPs, where cache contentions and memory access demands are escalated. Evaluations based on various workload mixes have demonstrated significant advantages of the CG-prefetcher over other existing prefetching schemes on CMPs.

Acknowledgements

This work was supported in part by NSF grant EIA-0073473 and by research and equipment donations from Intel Corp. We also thank anonymous referees for their helpful comments.

References

[1] Advanced Micro Devices. AMD Demonstrates Dual Core Leadership. http://www.amd.com, 2004.

[2] L. Barroso et al. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. In *Proc. of 27th Int'l Symp. on Computer Architecture*, pages 165-175, June 2000.

[3] S. Borkar. Microarchitecture and Design Challenges for Gigascale Integration. In *Proc. of 37th Int'l Symp. on Microarchitecture,* 1st Keynote, pages 3-3, Dec. 2004.

[4] P. Bose. Chip-Level Microarchitecture Trends. In *IEEE Micro*, Vol 24(2), page 5, Mar-Apr. 2004.

[5] R. Kalla, B. Sinharoy, and J. Tendler. IBM POWER5 Chip: A Dual-Core Multithreaded Processor. In *IEEE Micro*, Vol 24(2), pages 40-47, Mar-Apr. 2004.

[6] M. Charney and A. Reeves. Generalized correlation based hardware prefetching. *Technical Report EE-CEG-95-1,* Cornell University, Feb. 1995.

[7] Tien-Fu Chen, Jean-Loup Baer. Reducing memory latency via non-blocking and prefetching caches. In *Proc. of the 5th Int'l Conf. on ASPLOS,* pages 51-61, Oct. 1992.

[8] T. M. Chilimbi. On the stability of temporal data reference profiles. In *Int'l Conf. on PACT*, page 151, Sept. 2001.

[9] T. M. Chilimbi and M. Hirzel. Dynamic hot data stream prefetching for general-purpose programs. In *Proc. of the SIGPLAN '02 Conference on PLDI*, pages 199-209 June 2002.

[10] Z. Hu, M. Martonosi, S. Kaxiras. TCP: Tag Correlating Prefetchers. In *Proc. of 9th Ann Int'l Symp. on HPCA*, pages 317-326, Feb 2003.

[11] D. Joseph, and D. Grunwald. Prefetching Using Markov Predictors. In *Proc. of 26th Int'l Symp. on Computer architecture*, pages 252-263, Jun 1997.

[12] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. of 17th Int'l Symp. on Computer Architecture,* pages 364-373, May 1990.

[13] S. Kapil. UltraSPARC Gemini: Dual CPU Processor. *Hop Chips 15*, Aug. 2003.

[14] R. Kumar, N. P. Jouppi, and D. Tullsen. Conjoined-Core Chip Multiprocessing. In *Proc. of 37th Int'l Symp. on Microarchitecture*, pages 195-206, Dec 2004.

[15] S. Lacobovici, L. Spracklen, S. Kadambi, Y. Chou, and S. G. Abraham. Effective Stream-Based and Execution-Based Data Prefetching. In *Proc. Of the 18th ICS*, pages 1-11, June 2004.

[16] A. Lai, C. Fide, and B. Falsafi. Dead-block Prediction & Dead-block Correlating Prefetchers. In *Proc. of 28th Int'l Symp. on Computer Architecture*, pages 144-154, July 2001.

[17] P.S. Magnusson et al. Simics: A Full System Simulation Platform. In *IEEE Computer*, pages 50-58, Feb. 2002.

[18] T. Maruyama. SPARC64 VI: Fujitsu's Next Generation Processor. *Microprocessor Forum 2003*, Oct. 2003.

[19] C. McNairy and R. Bhatia. Montecito – The Next Product in the Itanium Processor Family. *Hot Chips 16*, Aug. 2004.

[20] K. Nesbit and J. Smith. Data cache prefetching using a global history buffer. In *Proc. of 10th Int'l Symp. on High Performance Computer Architecture,* pages 96-105, Feb. 2004.

[21] K. Olukotun. The Case for a Single-Chip Multiprocessor. In *Proc. of 7th Int'l Conf. on ASPLOS,* pages 2-11, Oct. 1996.

[22] S. Sair, and M. Charney. Memory Behavior of the SPEC-2000 Benchmark Suit. *Technical Report, IBM Corp.* Oct. 2000.

[23] A. Saulsbury, F. Dahlgren, and P. Stenstrom. Recency-based TLB preloading. In *Proc. of the 27th int'l symp. on Computer architecture*, pages 117-127, May 2000.

[24] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *Proc. of 10th Int'l Conf. on ASPLOS*, pages 45–57, Oct. 2002.

[25] G. Sohi. Single-Chip Multiprocessors: The Next Wave of Computer Architecture Innovation. In *Proc. of 37th Int'l Symp. on Microarchitecture,* 2nd Keynote, pages 143-143, Dec. 2004.

[26] Y. Solihin, J. Lee, and J. Torrellas. Using a user-level memory thread for correlation prefetching. In *Proc. of the 29th int'l symp. on Computer architecture*, pages171-182, May 2002.

[27] L. Spracklen and S. Abraham. Chip Multithreading: Opportunities and Challenges. In *Proc. of 11th Int'l Symp. on HPCA,* pages 248-252, Feb. 2005.

[28] L. Spracklen and Y. Chou. Effective Instruction Prefetching in Chip Multiprocessors for Modern Commercial Applications. In *Proc. of 11th Int'l Symp. on HPCA,* pages 225-236, Feb. 2005.

[29] S. P. Vanderwiel, and D. J. Lilja. Data Prefetch Mechanisms. *ACM Computing Surveys*, pages 174-199, June 2000.

[30] Z. Wang, and D. Burger, et al. Guided region prefetching: a cooperative hard-ware/software approach. In *Proc. of 30th Int'l Symp. on Computer Architecture*, pages 388-398, June 2003.

[31] T. Wenisch, S. Somogyi, et al. Temporal Streaming of Shared Memory. In *Proc. of 32nd Int'l Symp. on Computer Architecture,* pages 222-233, June 2005.

[32] Z. Zhu, and Z. Zhang. A Performance Comparison of DRAM Memory System Optimizations for SMT Processors. In *Proc. of 11th Int'l Symp. on HPCA,* pages 213- 224, Feb. 2005.