International Journal of Automation and Computing 10(6), December 2013, 587-596
DOI: 10.1007/s11633-013-0757-2

The Case of Using Multiple Streams in Streaming

Muhammad Abid Mughal Hai-Xia Wang Dong-Sheng Wang
Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science & Technology, Tsinghua University, Beijing 100084, China

Abstract: Off-chip replacement (capacity and conflict) and coherent read misses in a distributed shared memory system cause execution to stall for hundreds of cycles. These off-chip replacement and coherent read misses are recurring and forming sequences of two or more misses called streams. Prior streaming techniques ignored reordering of misses and not-recently-accessed streams while streaming data. In this paper, we present stream prefetcher design that can deal with both problems. Our stream prefetcher design utilizes stream waiting rooms to store not-recently-accessed streams. Stream waiting rooms help remove more off-chip misses. Using trace based simulations, our stream prefetcher design can remove 8% to 66% (on average 40%) and 17% to 63% (on average 39%) replacement and coherent read misses, respectively. Using cycle-accurate full-system simulation, our design gives speedups from 1.00 to 1.17 of princeton application repository for shared-memory computers (PARSEC) workloads running on a distributed shared memory system with the exception of dedup and swaptions workloads.

Keywords: Prefetching, stream first in first out (FIFO), princeton application repository for shared-memory computers (PARSEC), stream waiting rooms, reordering of misses, sequitur.

1 Introduction

The performance of a computer system depends not only on its processor but memory as well. For decades, the memory has been the main barrier for computer system performance. In the past decades, advancements in semiconductor fabrication technology along with micro-architectural and circuit innovations gave rise to increased processor speed, while during the same period, architects focused more on increasing the memory density than its speed, resulting in an ever growing processor-memory performance gap. However, with the introduction of chip multiprocessors (CMPs), processor clock frequencies are increasing less rapidly than in the past, yet bandwidth contention, interconnect latencies, etc. cause execution to stall for hundreds of cycles. Princeton application repository for shared-memory computers (PARSEC) workloads[1, 2] running on a distributed shared memory system stall for hundreds of cycles due to off-chip replacement (capacity and conflict) and coherent read misses[3]. Limited instruction window/reorder buffer (ROB) size[4] and data dependence (presented in PARSEC workloads) allow only a certain fraction of this off-chip read latency to be overlapped with computation. These off-chip replacement and coherent read misses are recurring and forming recurring sequences of two or more misses called streams[3]. The goal of this paper is to remove recurring off-chip replacement and coherent read misses, i.e., misses that follow address-correlation[5−7], using streaming (an act of early fetching data from memory using stream).

Prefetching is one of the ways to mitigate processor memory performance gap, also known as memory wall. Stride prefetchers[8] issue prefetches only when stride pattern is found. We found that most of the off-chip replacement and coherent read misses in most of the PARSEC workloads do not follow strided patterns.

Prior correlation-based prefetching approaches (temporal streaming (TS)[7], global history buffer[5], Markov prefetcher[6], etc.) locate single or multiple streams. Some of these techniques[7] compare multiple streams assuming that the order of addresses remains constant across stream recurrences. However, order of addresses is not always the same from one recurrence to another, called reordering. This reordering of addresses can be more common in CMPs where multiple on-chip processor cores compete for last-level shared cache. Fig. 1 shows reordering of off-chip coherent addresses from streamcluster workload. Similar examples can also be found for replacement misses. As Fig. 1 shows that misses are recurring but the order always does not remain the same, e.g., in one recurrence address 0x0c0048b1 follows 0x0c97bdc1 whereas in another recurrence 0x0c97bdc1 follows 0x0c0048b1. Moreover, we have observed using sequitur[9, 10] that streams are not unique and can start with the same address(es). In other words, streams share addresses. When stream recurs, most of the time it follows the most recently-accessed stream. However, sometimes stream does follow not-recently-accessed stream. Prior correlation-based prefetching approaches[5−7] do not utilize such streams while streaming data.

Based on above observations, we present hardware design called stream prefetcher (SP) that does not compare multiple streams while streaming data. Moreover, SP exploits a component, called stream waiting rooms (SWRs), to capture not-recently-accessed fetched streams. SWRs provide the most likely not-recently-accessed stream (to be followed) in case node is not following the most recently-accessed stream. Using a combination of trace-driven and cycle-accurate full-system simulation of a distributed shared memory system running PARSEC workloads, we show:

Manuscript received December 19, 2012; revised April 10, 2013
This work was supported by Higher Education Commission (Pakistan), National High Technology Research and Development Program of China (863 Program) (No. 2008AA01A201), Natural Science Foundation of China (Nos. 60833004 and 60970002), and TNList Cross-discipline Foundation.

1) Our SP design can eliminate 8 % to 66 % (on average 40 %) and 17 % to 63 % (on average 39 %) replacement and coherent read misses, respectively.

2) Using SWRs, our SP design can remove more off-chip misses.

3) The performance results are not constant across SP and competitive prefetchers. For some workloads, SP excels whereas for others competitive prefetchers.

4) Our SP design gives speedups from 1.00 to 1.17 with the exception of dedup and swaptions.

. . . 0x000048b1 0x0cc591b1 0x001048b1 0x008c9ca1 0x088c9ca1 ***0x0c97bdc1***
0x0c0048b1 0x000048b1 0x0cc591b1 0x001048b1 0x00b24fc1 0x008c9ca1 0x088c9ca1
0x0c0048b1 0x0c0048b1 ***0x0c0048b1 0x0c97bdc1*** 0x000048b1 0x001048b1
0x00b23fc1 0x008c9ca1 0x088c9ca1 ***0x0c0048b1 0x0c97bdc1*** 0x0c0048b1 0x0c0048b1
0x0c0048b1 0x000048b1 0x0cc591b1 0x001048b1 . . .

Fig. 1 Addresses of cache lines, in hexadecimal format, which incurred off-chip coherent read misses. These addresses are taken from streamcluster workload. This shows that misses are recurring but order may vary from one recurrence to another

The rest of the paper is organized as follows. Section 2 discusses classification of off-chip misses. Section 3 elaborates our SP design. Section 4 describes simulated system and workloads. Section 5 presents results, comparisons with other competitive prefetchers and performance. Finally, Section 6 concludes the paper.

2 Off-chip misses and streams

We classify off-chip read miss into three types:

1) Cold: Very first read access to a line that is missed in L2 cache is classified as cold miss.

2) Replacement: Second or subsequent read access to a line that missed in L2 cache is classified as replacement miss. These misses occur because of the L2 cache capacity or conflicts.

3) Coherent: First read access to a line that is written by another node is classified as coherent miss. We do not differentiate between true and false coherent read misses because we are interested in coherent traffic occurred with real system parameters, e.g., cache line size[11].

With the help of sequitur, we found that off-chip replacement and coherent read misses are recurring, i.e., they keep repeating[3, 12]. A repetitive sequence of two or more misses is called stream. Coherent read misses form coherent streams when read-write shared data structures with repetitive traversals are transferred among nodes. Replacement misses form replacement streams when a repetitive traversal accesses a data structure that exceeds the L2 caches capacity or traversals are separated by many intervening accesses. Even though off-chip replacement and coherent read misses are recurring, but some reordering of misses is present from one recurrence to another. This reordering implies that multiple streams should not be compared while streaming data. Therefore, our SP utilizes one stream while streaming data. We use term *off-chip read misses* to imply both off-chip replacement and coherent read misses. Similarly, we use term *stream* to imply both replacement and coherent streams.

3 Stream prefetcher

Each node has one stream prefetcher. Components that belong to SP are highlighted in Fig. 2. SP streams data for both off-chip replacement and coherent read misses. With the help of directory embedded in each node, SP dynamically identifies off-chip read misses as replacement or coherent. It stores replacement and coherent read misses into separate off-chip circular buffers as these misses can grow to a large number. Circular buffers that store off-chip coherent and replacement read misses are called coherent pointer buffer (CPB) and replacement pointer buffer (RPB), respectively. A part of memory is used to implement CPB and RPB. These buffers are not visible to operating system. As each processor executes instructions speculatively, SP stores read misses into corresponding CPB/RPB only when corresponding read instructions get committed. This makes sure that both CPB and RPB store read misses from correct execution path. Like temporal streaming of shared memory[7] and prior proposals for recording on-chip generated metadata in memory (e.g., [13]), SP separately packetizes off-chip replacement and coherent addresses in the form of cache blocks and ships them off-chip to the corresponding CPB/RPB. To locate multiple streams across the system, SP extends each directory entry with multiple separate CPB and RPB pointers and CP and RP fields. Each CPB/RPB pointer in the directory includes a node ID and an offset within the CPB/RPB where the miss is stored. The storage overhead is (number of CPB pointers)$\times$($\log_2$(nodes) + $\log_2$(CPB size)) + (number of RPB pointers)$\times$($\log_2$(nodes) + $\log_2$(RPB size)) bits.

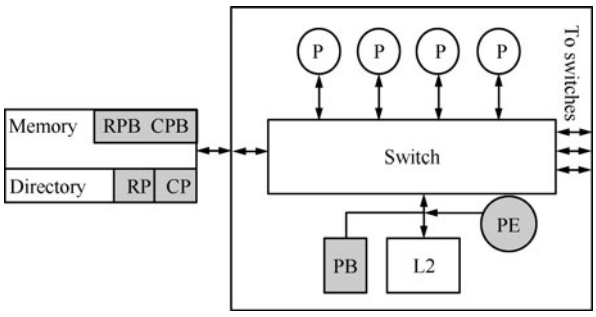


Fig. 2 Node along with stream prefetcher. Highlighted components are new and added for stream prefetcher. Each node has four cores and a shared L2 cache

3.1 Storing read misses into off-chip buffers

Fig. 3 shows actions involved in storing off-chip coherent/replacement read misses into the corresponding off-chip buffer, CPB/RPB. 1) Node issues a read request to memory/directory node when off-chip miss occurs. 2) Directory annotates data response to classify miss as either coherent or replacement. This annotation avoids sending separate message for the classification of miss. 3) Node gets data. SP waits for corresponding instruction retirement that caused miss. If that instruction gets retired, SP packetizes that miss and issues packet to CPB/RPB if packet size has been reached. 4) Finally, CPB/RPB controller receives packet

and starts extracting miss addresses from a packet. It inserts each address into CPB/RPB and then issues a message to home directory node to update CP/RP field. Writing addresses to CPB/RPB and updating CP/RP field take place in background and are off the critical path of the processor.

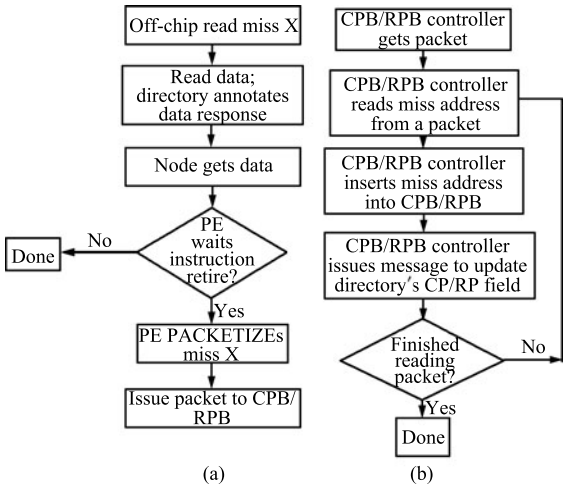


Fig. 3 Actions involved in storing a read miss to off-chip buffer

3.2 Locating and forwarding stream

With the help of CP/RP field(s), SP locates stream(s). Fig. 4 shows actions involved in locating and forwarding a stream. 1) Node incurs off-chip read miss and issues read request to memory/directory node. 2) Directory node detects if miss is a replacement or coherent. If miss is either replacement or coherent, then it reads RP or CP field and issues read stream message to corresponding node. 3) CPB/RPB controller sends stream, reads from CPB/RPB, to a node where miss occurred. 4) Finally, node gets stream and inserts it to either stream FIFO or SWR. Stream read access latency is high, but it is either compensated (on the initial miss) or overlapped through streaming look-ahead.

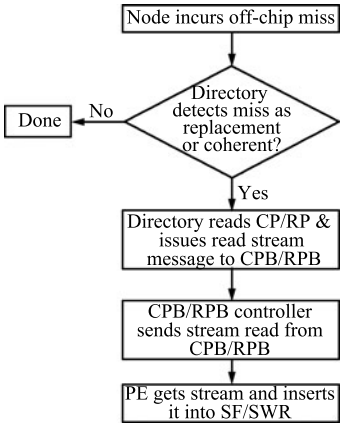


Fig. 4 Actions involved in reading a stream from off-chip buffer

3.3 Prefetch engine

Fig. 5 depicts heart of SP, called prefetch engine (PE). Its design is motivated by two main observations: 1) Reordering of misses and 2) capturing of not-recently-accessed streams. To deal with these observations, PE streams data uses single stream and utilizes SWRs to capture multiple not-recently-accessed streams. PE has three components, namely, stream FIFOs (SFs), stream waiting room (SWRs) and issue prefetch engine (IPE).

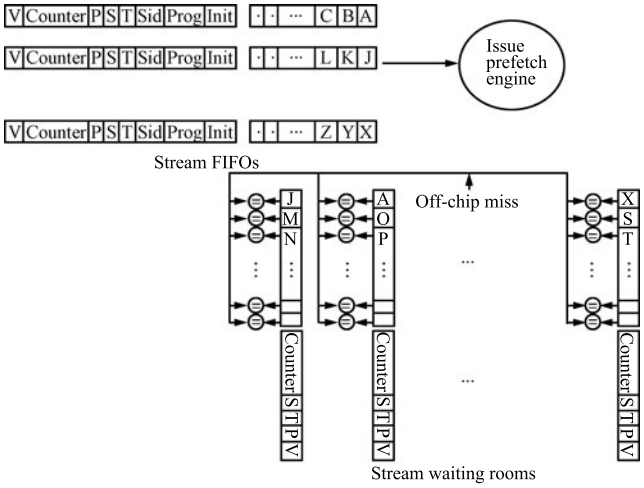


Fig. 5 Prefetch Engine which manages fetched streams and issues prefetch

Each SF is composed of one FIFO, 10-bit stream identity register (Sid), 1-bit init flag register (Init), 1-bit progress flag register (Prog), 1-bit stream type register (ST), pointer register (P), 4-bit saturate counter (Counter), and 1-bit valid register (V), as shown in Fig. 5. Each FIFO stores fetched stream. PE dynamically assigns identity to each stream, and saved in Sid. Using ten bits for stream identity makes sure that multiple active streams may not have same Sid. Init bit tells PE whether SF is just initialized or not. Prog bit tells PE if stream is still helpful avoiding off-chip misses. PE increments saturate counter whenever demand requests hit in prefetch buffer (PB). Saturate counter determines which streams are more useful in terms of removing off-chip misses. Saturate counter is used by IPE and SF replacement policy. Replacement policy gives preference to SFs with lower values of saturate counter. To reuse SFs that outlived their usefulness but remained in there because of high saturate counter value, saturate counter is decremented for every ten SF allocation requests. ST bit tells about the type of stream that is either replacement or coherent. ST is used in conjunction with pointer register and directs PE which buffer to read more stream, i.e., CPB or RPB. Valid bit (V) tells whether SF is valid or not.

Each SWR is composed of buffer, comparators, 3-bit saturate counter, 1-bit stream type register (ST), pointer register (P) and 1-bit valid register (V), as shown in Fig. 5. ST bit, P register, and V bit have the same goals as in SF. For each bit in the buffer, there is a comparator. Whenever there is an off-chip miss, all SWRs are searched against miss in parallel for a match. If a match is found, then saturate counter of corresponding SWR is incremented. Later PE selects SWR having the highest value of counter to be moved to SF as it is likely that node will follow that stream. PE moves buffer contents (after matching point), ST, and P register from selected SWR to SF. Later, that chosen SWR is reset. SWRs are replaced using least recently used policy. The purpose of SWRs is to capture mul-

tiple not-recently-accessed streams starting with the same miss. Streams are not unique and same read miss(es) can belong to more than one stream[3]. SWRs provide the most likely not-recently-accessed stream in case node is not following the most recently-accessed stream. This is the main motivation of using SWRs. As we will see in Section 5, using SWRs helps remove more off-chip misses.

IPE is a counter based issue-prefetch engine that issues prefetches based on SFs saturate counter. IPE gives more priority to SFs having the higher value of saturate counter i.e., first it issues prefetches from SF having the highest value of counter then the second highest and so on. Each PE has number of SFs and SWRs but only one IPE. PE always locates multiple streams starting with the same miss using multiple CP/RP fields. However, only most recently seen stream goes to SF and the rest of the streams stay in the SWRs.

Fig. 6 shows the structure of the prefetch buffer, which is used to store prefetched data. PB is implemented as a fully-associative buffer and accessed in parallel with L2 cache. For each node, one PB is used. Using separate storage for prefetched data helps avoiding pollution of L2 cache when streams are useless and modification to baseline L2 cache. Each PB entry includes a valid bit, address, data, state, sharers and the identity of the stream to which it belongs to. Sharers keep track of number of on-chip processors where this line is present. State is the cache coherence protocol state. The data in PB can be dirty. Since each node has number of processors, maintaining clean data in PB is not a trivial task. We use same baseline cache coherence protocol to maintain the coherence in PB. PB entries are replaced using a least recently used (LRU) policy.

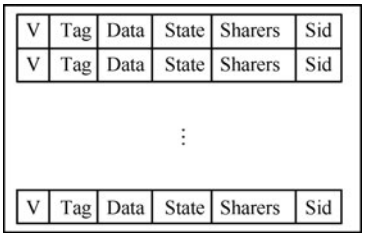


Fig. 6 Fully-associative prefetch buffer that holds prefetched lines. Prefetches are only allocated to this buffer

3.4 SP working

Like temporal streaming[7], SP locates multiple streams following procedure described in Section 3.2 whenever off-chip miss occurs. All fetched streams go to SWRs except the most recently seen stream which goes to SF. When stream is written to SF, PE sets SF′s Init bit so that IPE can issue prefetch requests from it upon its turn. After issuing prefetch requests, PE resets its Init bit. IPE does not issue more prefetch requests unless its Prog bit is set. Later, when processor read accesses hit in the PB, the PE is notified to set Prog bit and increment saturate counter of the corresponding stream identified by Sid. Again, IPE issues prefetch requests from this stream upon its turn and resets Prog bit. Setting Prog bit each time whenever demand read request hits in a PB is a way to signal useful streams to IPE (useful streams help removing off-chip misses whereas useless streams do not, and lead to increased number of prefetches that are not used by processor). This setting and resetting of Prog bit is critical as it signals IPE that this stream is still useful or not. If PE finds that stream is useful and its FIFO is $\frac{3}{4}$-th empty then it reads more stream addresses from off-chip buffer using pointer register (P) and ST bit. If neither Init nor Prog bit is set, IPE issues no prefetch requests. In case node is not following most recently-accessed stream, it will incur off-chip miss. PE compares that miss against all SWRs and ultimately SWRs can provide stream (to SFs) likely to be followed by node. Later, IPE can issue prefetch requests using that stream as described above.

4 Methodology

4.1 Target system and workloads

We simulate a 4-node distributed shared memory system with 4 GB memory running Sun Solaris 10 operating system. Each node has four processors, 8-way 2 MB unified shared L2 cache, PB, and 1 GB memory, as shown in Fig. 2. Each processor has 2-way 16 KB L1 instruction and data caches. Running number of simulations with varying L2 cache reveals that 2 MB L2 cache is sufficient for most of the PARSEC workloads to capture temporal locality in the data footprints of application. We configured Simics[14] to model UltraSparc III plus target system. We collect our results using a combination of trace-driven and cycle-accurate full system simulation of a distributed shared-memory system using GEMS. Multifacet general execution-driven multiprocessor simulator (GEMS) toolset[15] is a simulation framework that uses modular component-based design. A GEMS builds on Wind River Simics, a full system simulator that allows functional emulation of unmodified applications and operating systems. GEMS furnishes simics with cycle-accurate models of an out-of-order processor core, single/multi-banked cache hierarchies, broad range of coherence protocols, distributed memory and interconnect networks. Using table driven specification methodology GEMS accurately models state transitions (including transient states) of the coherence protocols in cache and memory controllers.

Table 1 lists the PARSEC workloads we use for our study. All workloads use sim small data set. We chose PARSEC because of its diversity, emerging workloads and state-of-art algorithms[2]. All workloads were compiled using GCC 3.4.3 with -o3 optimization. Our trace-based analysis use memory access traces collected from GEMS with in-order execution, no memory system stalls, and a fixed Instruction Per Cycle (IPC) of 1.0. We fast forwarded the simulation until the beginning of region of interest (ROI)[1] before collecting traces. ROI is the parallel phase of each PARSEC workload. Moreover, all prefetch instructions are filtered out while collecting traces. Each trace based simulation uses these traces and we perform measurements for the whole ROI. Our cycle-accurate full system simulations use out-of-order processor cores. Each processor core can fetch, decode, dispatch, execute and retire eight instructions per

cycle. Other relevant parameters of target system are shown in Table 2. Because of the variability[16] in multi-threaded workloads, we use total execution time as a performance metric. Speedup is the ratio of total execution time of base system and enhanced system. We also show average opportunity and average coverage that are the averages across PARSEC workloads.

5 Results

5.1 Opportunity

The maximum number of off-chip read misses that can be removed is called opportunity. Fig. 7 shows opportunity for replacement and coherent read misses. Sequitur calculates this opportunity by adding all recurring misses. Average opportunity is the average of all PARSEC workloads opportunities, which is 53% for replacement and 68% for coherent read misses. Coverage is the fraction of all read misses that SP correctly predicts and eliminates. Average coverage is the average of all PARSEC workloads coverage. Fig. 8 shows coverage for replacement and coherent read misses for a stream look-ahead of eight cache blocks along with number of fetched streams and no SP hardware restrictions (unlimited CPB/RPB, unlimited PB, two SFs and eight SWRs). Using infinite number of SFs leads to a delay in issuing prefetches from useful streams as streams that outlived their usefulness remain stayed in SFs. Moreover, using infinite SWRs causes streams to stay forever in SWRs that may lead useless streams to be moved to SFs. In Fig. 8, Stream1 implies that SP locates only one most recently-accessed stream, Stream2 implies that SP locates two most recently-accessed streams and so on. SWRs are not used when SP locates only one stream. The coverage increases as the number of fetched streams is increased. This increase is due to SWRs that move likely not-recently-accessed streams to SFs. When SP locates two streams instead of one, the coverage increases for all workloads, with maximum increase in swaptions (8% for replacement and 13% for coherent misses) while average coverage increases 3% and 4% for replacement and coherent misses, respectively. When the number of streams increased from two to four or eight, the increase in average coverage is not appreciable (less than 2%), so we configure SP to locate two most recently-accessed streams. The results hereafter assume two streams.

5.2 Sensitivity

So far, we have used two SFs, eight SWRs, infinite PB, unlimited CPB/RPB, and eight cache blocks stream look-ahead. For all following results, SP locates two streams.

5.2.1 Stream FiFOs, stream waiting rooms and prefetch buffer

We run a number of experiments with varying number of SFs to see their effect on coverage (Fig. 9). For most of the workloads, increasing number of SFs from two to eight decreases average coverage for replacement misses by 2% whereas average coverage for coherent misses increases by 3%. Therefore, increasing number of SFs gives conflicting behavior. To remove this conflict, SP uses separate exclusive number of SFs: Two SFs for replacement streams and eight SFs for coherent streams as such SFs count provides maximum coverage for each type of misses (Fig. 9). The rest of the results use separate exclusive SFs count for replacement and coherent streams.

Up to now we have used eight SWRs because SP locates maximum eight streams and seven of them go to SWRs. Fig. 10 shows SWRs effect on coverage. When SWRs are reduced to four, the average coverage decreases only less than 1% for replacement and coherent misses, so we configure SP to use four SWRs.

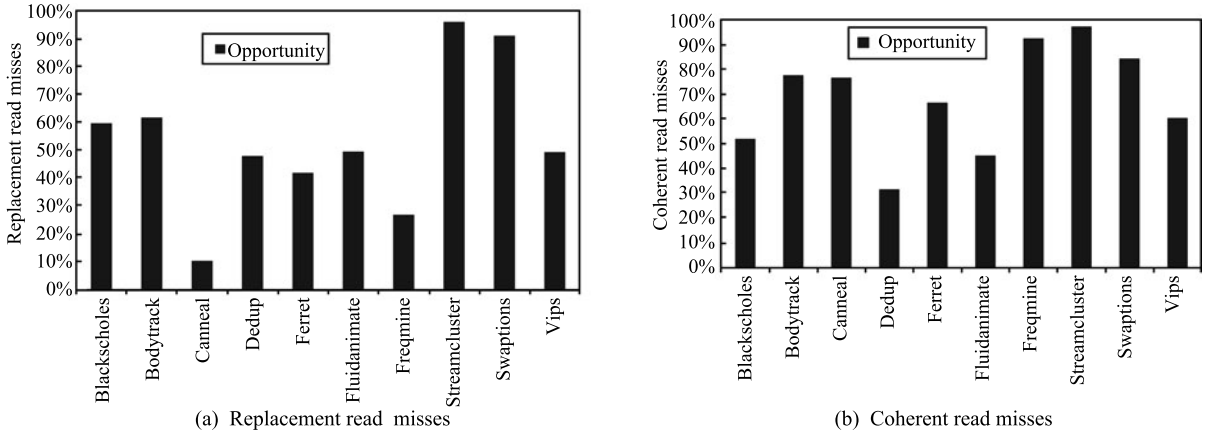
Running number of experiments with varying PB size reveals that for coherent misses, even 1 KB PB provides same

Table 1 Workloads

Workload	Description
Blackscholes	Calculates the prices for a portfolio analytically with the black-scholes partial differential equation (PDE); financial analysis applications
Bodytrack	Tracks a human body with multiple cameras through an image sequence; computer vision applications
Canneal	Electronic design automation (EDA) kernel for minimizing the routing cost of a chip design; engineering applications
Dedup	Pipelined compression kernel; enterprise storage applications
Ferret	Image search engine; similarity search applications
Fluidanimate	Simulates the underlying physics of fluid motion; animation applications
Freqmine	Identifies frequently occurring patterns in a transaction database; data mining applications
Streamcluster	Computes an approximation for the optimal clustering of a stream of data points; data mining applications
Swaptions	Prices a portfolio of swaptions with the heath-jarrow-morton framework; financial analysis applications
VIPs	Applies a series of transformations to an image; media applications

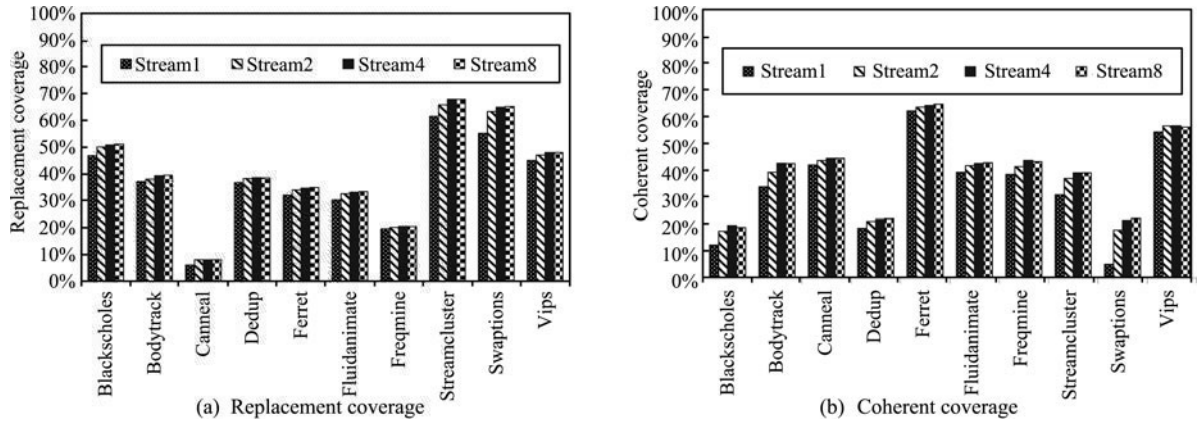
Table 2 System specification

System	Specification
Processor	UltraSPARC III plus ISA; 11-stage pipeline; out-of-order execution; 8-wide dispatch/retirement; 256-entry ROB; 64-entry LSQ and writebuffer
L1 caches	Split, 2-way 16 KB L1 instruction and data caches; 2 cycles hit latency; 32 MSHRs
L2 cache	Unified, 8-way 2 MB; 7 cycles hit latency; 128 MSHRs
Memory	Total 4 GB, 1 GB/Node; 300 cycles access latency
Protocol	MOSI, 64 B coherence unit
Interconnect	Full mesh; off-chip 80 cycles latency per hop; 64 bytes wide links
PB	16 KB fully-associative cache; 1 cycle hit latency



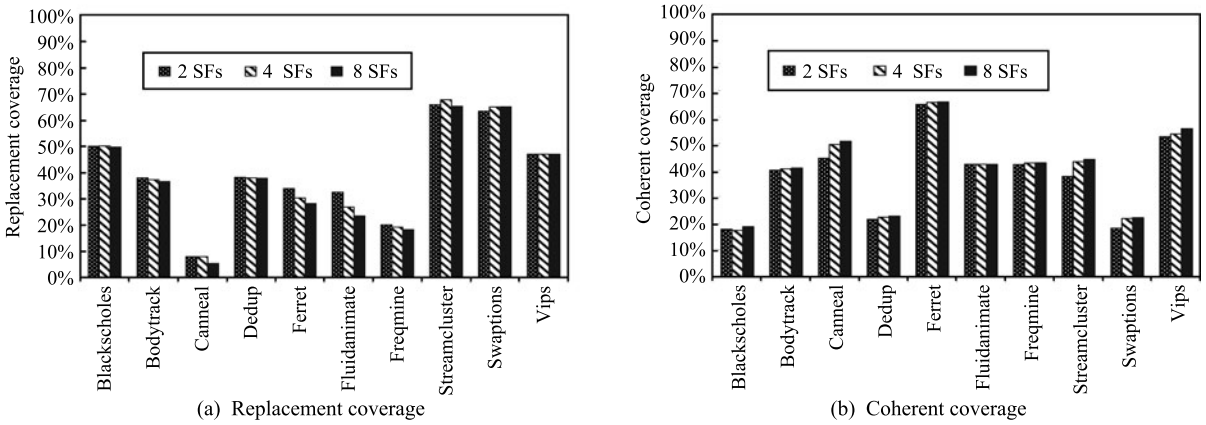
(a) Replacement read misses (b) Coherent read misses

Fig. 7 Opportunity



(a) Replacement coverage (b) Coherent coverage

Fig. 8 Coverage



(a) Replacement coverage (b) Coherent coverage

Fig. 9 Stream FIFO's effect on coverage

coverage as infinite PB, whereas for replacement misses a big-sized 16 KB PB provides same coverage as infinite PB (Fig. 11). When PB is increased from 1 KB to 16 KB, the average coverage increases by 21% and 0.1% for replacement and coherent misses, respectively. Small size PB for coherent misses implies that they are closer to each other in time than replacement misses. Replacement/coherent misses, that form replacement/coherent streams, do not necessarily mean that they also occurred close in time. Since each node has one shared PB, we use 16 KB PB.

5.2.2 Discards, off-chip buffers and bandwidth

Fig. 12 shows how stream look-ahead affects discards, presented as a fraction of misses in the base system. Discards are fetched cache blocks that are not used by processor. In Fig. 12, 4_8 means stream look-ahead of four cache blocks for replacement misses and eight for coherent misses. Using stream look-ahead of four provides reduced fraction of discards while coverage remains same or increases for replacement misses whereas it decreases for coherent misses slightly with the exception of canneal (figure not shown here). Rest of the results assume stream look-ahead of four

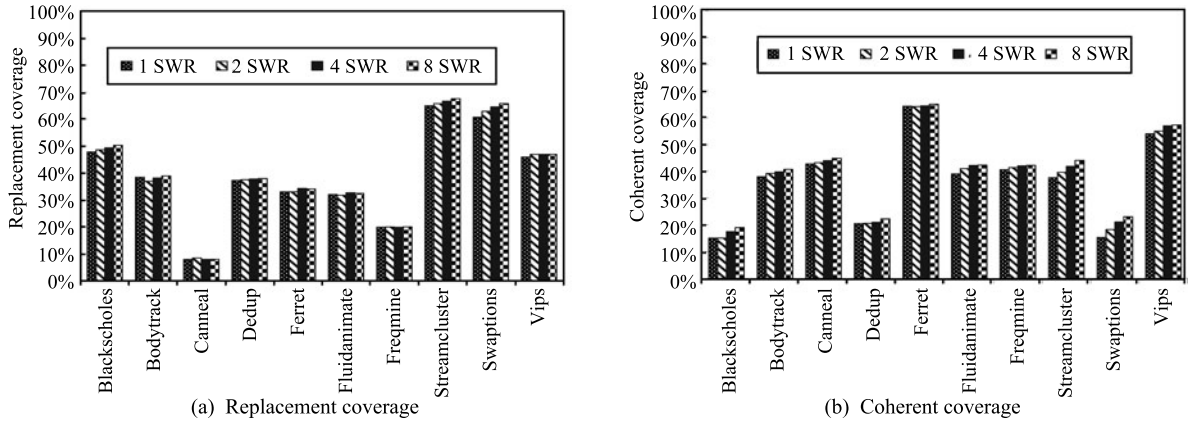


Fig. 10 Stream waiting room's effect on coverage

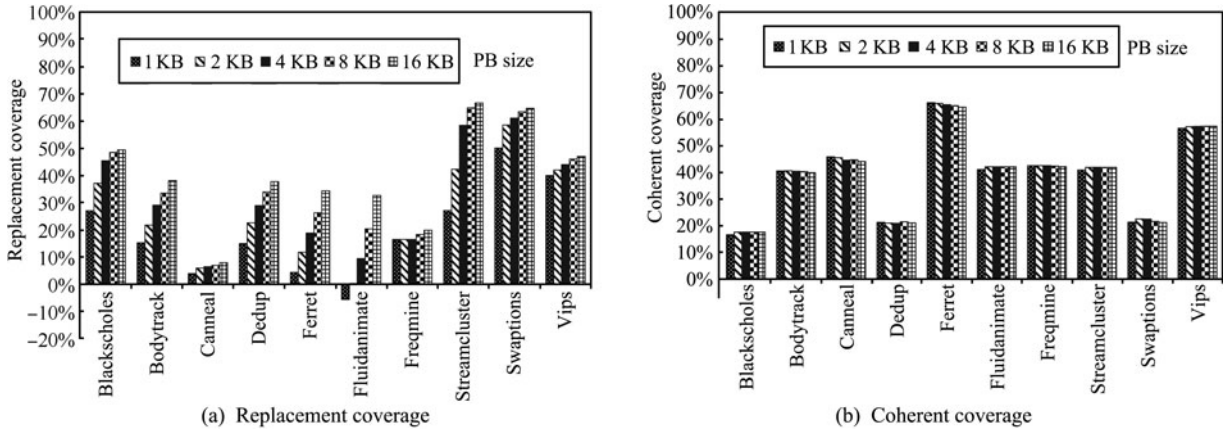


Fig. 11 Prefetch buffer's size effect on coverage

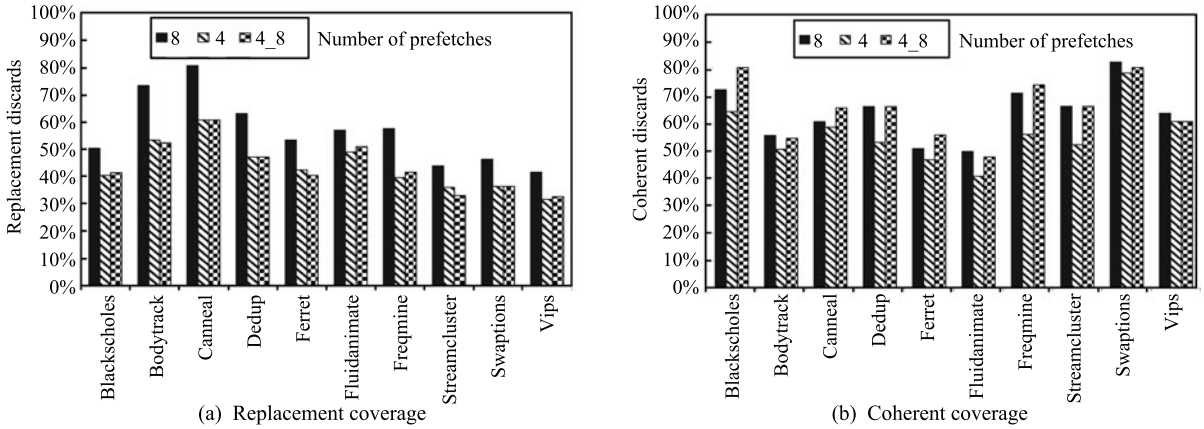


Fig. 12 Discards

cache blocks.

Off-chip buffers, CPB/RPB, must be big enough to hold useful streams. Once CPB/RPB gets full, overwriting to these buffers may cause to lose useful streams and hence coverage starts decreasing. We choose buffers sizes by instrumenting our simulator and use same size for both buffers. We vary both buffers sizes in the same simulation. Varying RPB size reveals that replacement coverage starts decreasing for freqmine and streamcluster when RPB size changed from 4 MB to 2 MB whereas it stays the same for other workloads, so we use 4 MB RPB (Fig. 13 (a)). In the same way, varying CPB size reveals that the coherent coverage starts decreasing for streamcluster and VIPs when CPB size changed from 1MB to 512KB whereas it stays the same for other workloads, so we use 1MB CPB (Fig. 13 (b)).

Fig. 14 shows bandwidth consumed by our design. The

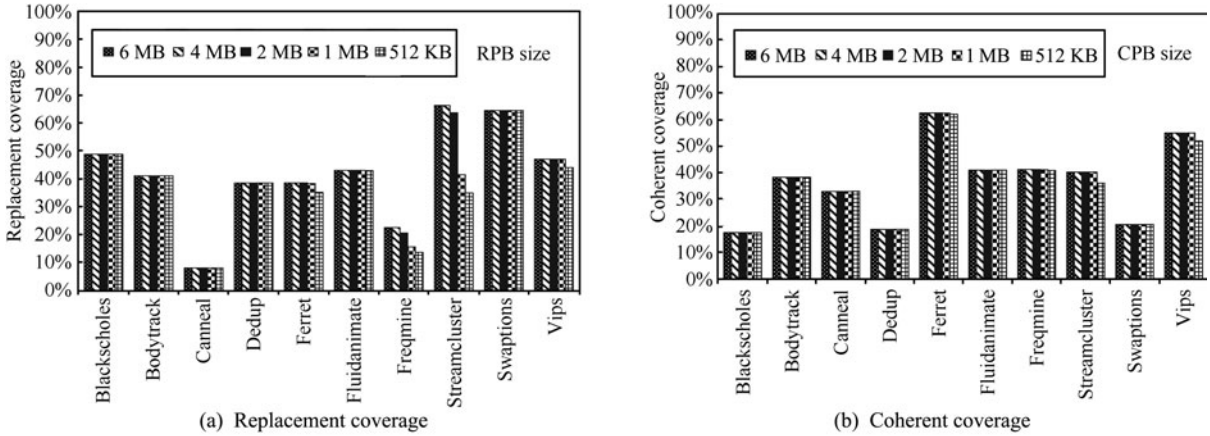


Fig. 13 Off-chip buffer's size effect on coverage

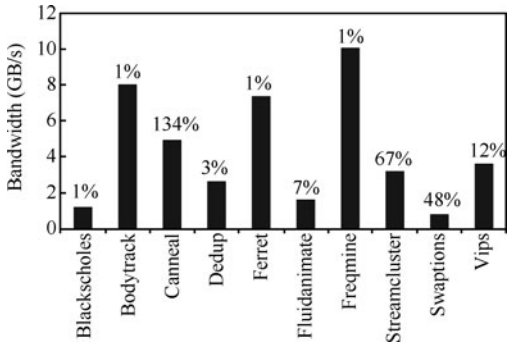


Fig. 14 Bandwidth comsumed by our design

annotation above each bar indicates the increase in bandwidth comparing to base system. This increase in bandwidth is mostly due to discards. Canneal has the highest increase in bandwidth as most of the replacement misses do not recur and hence result in many discards.

5.3 Comparison and performance

To see the effectiveness of SP in eliminating replacement and coherent read misses, we compare it against temporal streaming (TS), stride prefetcher (Strided)[8], and stream chaining (SC)[17] as shown in Fig. 15. TS has been shown as the best prefetcher for streams. In addition, we use stride prefetcher because it can be found in modern processors (e.g., Intel xeon, Intel i5/i7 desktop processor, AMD Opteron). We implement two-delta stride prefetcher[8] and use separate 512×4 set-associative caches for replacement and coherent read misses. We also implement SC[17] because it has been shown best for single-threaded applications.

Replacement read misses: SP outperforms all competitive prefetchers, with the exception of freqmine workload. The maximum coverage given by SP, TS, Strided, and SC is about 66% (for streamcluster), 25% (for streamcluster), 61% (for freqmine), and 36% (for freqmine), respectively. Stride prefetcher and SC outperform SP for freqmine whereas for bodytrack and dedup, stride prefetcher is comparable to SP. Average coverages given by SP, TS, Strided, and SC are 42%, 16%, 20%, and 14%, respectively.

Coherent read misses: For most of the workloads, SP performs as well as SC, but SC outperforms SP for dedup, fluidanimate, and VIPs. The maximum coverage given by SP, TS, Strided, and SC is about 62% (for ferret), 44% (for ferret), 50% (for dedup), and 67% (for VIPs), respectively. Stride prefetcher outperforms SP for dedup. Average coverages given by SP, TS, Strided, and SC are 37%, 14%, 16%, and 43%, respectively. So SC, on average, performs better than all other competitive prefetchers.

Unlike TS, stride prefetcher, and SC, SP produces more discards (Fig. 16) and hence consumes more bandwidth. Using cycle accurate full-system simulation of a 4-node distributed shared memory system, we measure the speedup of SP, TS, stride prefetcher, and SC as shown in Fig. 17. For some workloads, SP excels whereas for others, competitive prefetchers excel. SP gives best performance to blackscholes, ferret, fluidanimate, and streamcluster. TS gives best performance only to swaptions. Strided gives best performance to bodytrack, dedup, freqmine, and vips. SC gives best performance to only canneal. SP gives maximum performance to fluidanimate workload, 17.7%, whereas it decreases performance of dedup and swaptions.

6 Conclusions

This paper presented a stream prefetcher (SP) design that tries to remove recurring off-chip replacement (capacity and conflict) and coherent read misses. SP streams data using only single stream due to the reordering of misses. It introduces the concept of stream waiting rooms that capture not-recently-accessed streams. Stream waiting rooms provide likely not-recently-accessed streams to stream FIFOs in case future streams do not follow most recently-accessed stream. Results showed that, with two streams being fetched, stream waiting rooms help improve average coverage by 3% and 4% for off-chip replacement and coherent read misses, respectively. We showed using trace based simulation that SP can eliminate, on average, 40% and 39% replacement and coherent read misses, respectively. Using cycle-accurate full-system simulation, our design gives speedups from 1.00 to 1.17 for PARSEC workloads running on a 4-node distributed shared memory system with the exception of dedup and swaptions, where it degrades performance.

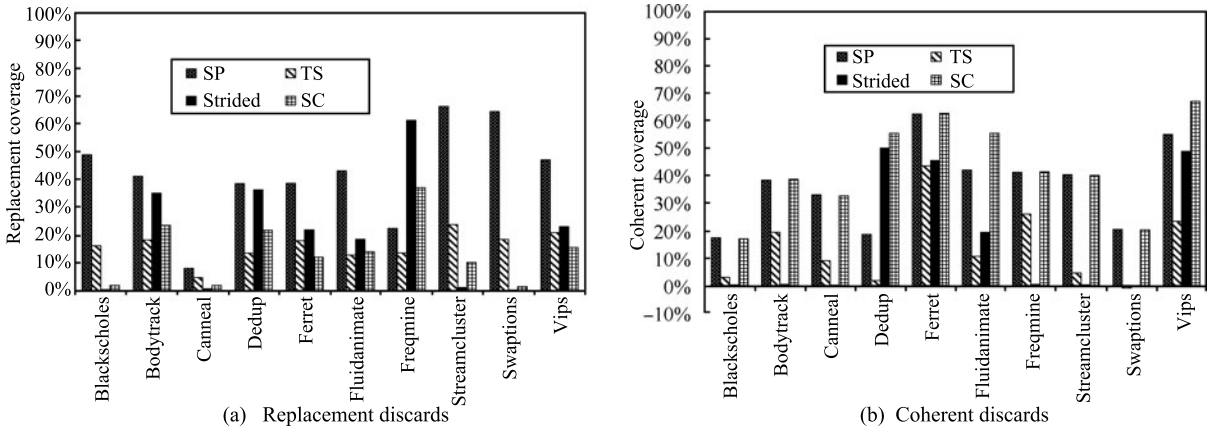


Fig. 15 Comparison of SP′s coverage

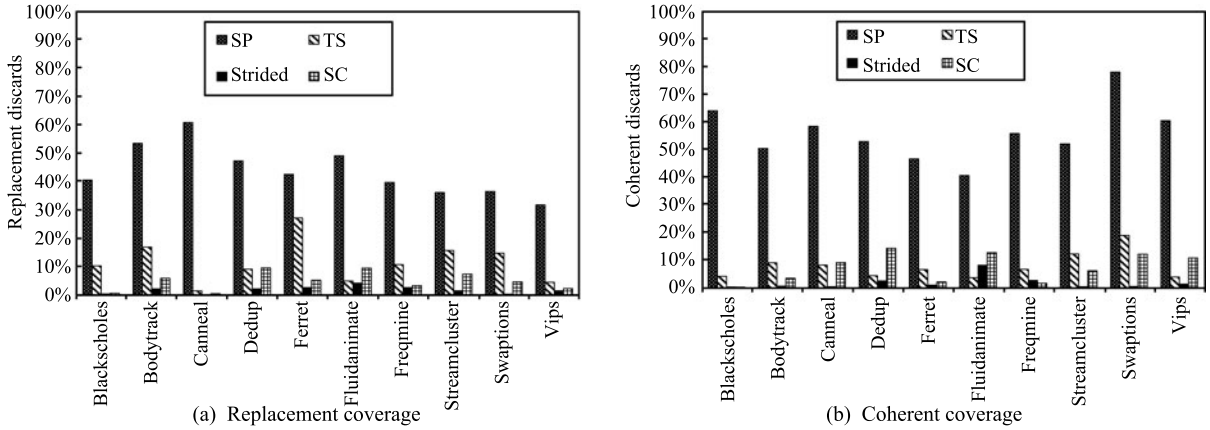


Fig. 16 Comparison of SP′s discards

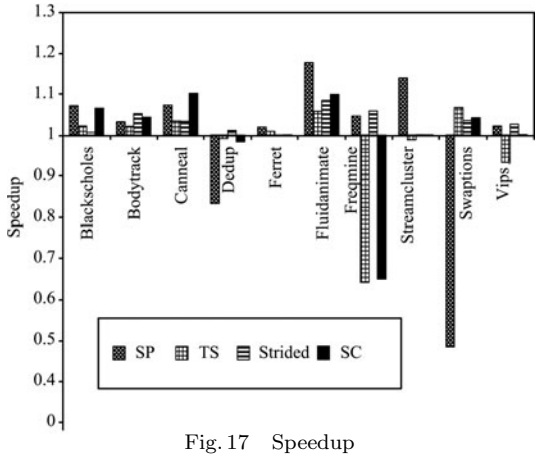


Fig. 17 Speedup

Acknowledgement

The authors would like to thank the anonymous reviewers for their insightful comments.

References

[1] C. Bienia. Benchmarking Modern Multiprocessors, Ph. D. dissertation, Princeton University, USA, 2011.

[2] C. Bienia, S. Kumar, J. P. Singh, K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ACM, New York, USA, pp. 72–81, 2008.

[3] M. A. Mughal, H. X. Wang, D. S. Wang. Analyzing off-chip misses in PARSEC for prefetching. *International Journal of Advancements in Computing Technology*, vol. 3, no. 10, pp. 299–306, 2011.

[4] O. Mutlu. Efficient Runahead Execution Processors, Ph. D. dissertation, The University of Texas, USA, 2006.

[5] K. J. Nesbit, J. E. Smith. Data cache prefetching using a global history buffer. In *Proceedings of 10th Symposium on High-performance Computer Architecture*, IEEE, Washington, DC, USA, pp. 96–105, 2004.

[6] D. Joseph, D. Grunwald. Prefetching using Markov predictors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, ACM, New York, USA, pp. 255–263, 1997.

[7] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamak, B. Falsafi. Temporal streaming of shared memory. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, IEEE, Madison, Wisconsin, USA, pp. 222–233, 2005.

[8] Y. Sazeides, J. E. Smith. The predictability of data values. In *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE, Research Triangle Park, NC, USA, pp. 248–258, 1997.

[9] C. G. Nevill-Manning, I. H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, vol. 7, no. 1, pp. 67–82, 1997.

[10] Sequitur, [online], Available: http://sequitur.info/.

[11] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, pp. 24–36, 1995.

[12] M. A. Mughal, H. X. Wang, D. S. Wang. Coherent temporal streams in PARSEC. In *Proceedings of the 6th International Conference on Networking, Architecture, and Storage*, IEEE, Dalian, China, pp. 295–301, 2011.

[13] C. Gniady, B. Falsafi. Speculative sequential consistency with little custom storage. In *Proceedings of the 10th International Conference on Parallel Architectures and Compilation Techniques*, IEEE, Barcelona, Spain, pp. 179–188, 2002.

[14] Windriver, [online], Available: http://www.windriver.com/.

[15] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R Alameldeen, K. E. Moore, M. D. Hill, D. A. Wood. Multifacets general execution-driven multiprocessor simulator (GEMS) toolset. *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

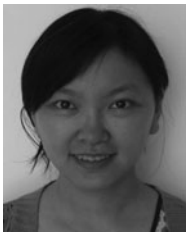
[16] A. R. Alameldeen, D. A. Wood. Variability in architectural simulations of multi-threaded workloads. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, IEEE, Anaheim, California, USA, pp. 7–18, 2003.

[17] P. Daz, M. Cintra. Stream chaining: Exploiting multiple levels of correlation in data prefetching. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ACM, New York, USA, pp. 81–92, 2009.

Muhammad Abid Mughal graduated from University of Sindh, Pakistan in 2001. He received his M. Sc. degree from Pakistan Institute of Engineering and Applied Sciences (PIEAS), Pakistan in 2005 and his Ph. D. degree from Tsinghua University, China in 2012. He is currently working as a senior scientist at PINSTECH, PAEC, Pakistan.

His research interests include chip multiprocessors and its memory subsystem design.

E-mail: mabidm_pieas@yahoo.com (Corresponding author)



Hai-Xia Wang graduated from Nankai University, China in 1998. She received her Ph. D. degree from Institute of Computing Technology, Chinese Academy of Sciences, China in 2004. She is currently an associate professor at the CPU and SoC Technology Research Center, Tsinghua University, China.

Her research interests include multi-core architecture and formal verification.

E-mail: hx-wang@mail.tsinghua.edu.cn

Dong-Sheng Wang received his B. Eng., M. Eng., and Ph. D. degrees in computer science from Harbin Institute of Technology, China in 1989, 1992 and 1995, respectively. He is now professor of the Department of Computer Science at Tsinghua University and the director of CPU and SoC Technology Research Center, Tsinghua University.

His research interests include computer architecture, multicore and SoC, disaster recovery, and high availability computing.

E-mail: wds@mail.tsinghua.edu.cn