Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications Workshops

Evaluating the Memory System Performance of Software-Initiated Inter-Core LLC Prepushing

Min Cai, Zhimin Gu
School of Computer Science and Technology
Beijing Institute of Technology
Beijing, China
Email: min.cai.china@gmail.com

***Abstract*—Data prefetching speculatively issue memory requests for data needed later by the main computation, and therefore can lead to increased stress on limited resources on chip multiprocessors. If not properly used, it can cause harmful effects such as cache pollution and waste of bandwidth. Therefore, accurate and fine grain measurement of the related runtime metrics is important as the first step in reducing harmful prefetches and increasing memory level parallelism on chip multiprocessors. However, the required measurement is prohibitively impossible on real machines without bringing nontrivial performance overhead and thus leading to inaccurate results.**

In this paper, we use cycle accurate full-system simulation to study the memory system performance of our previous proposed data prefetching technique with control of harmful prefetches on chip multiprocessors - software-initiated inter-core LLC prepushing. We modified the GEMS multiprocessor simulator to support trace-based measurement and offline analysis of MLP, DRAM BLP and their relationship with software-initiated inter-core LLC prepushing. Results show that, prepushing can achieve speedups of 1.628, 1.019 and 1.032 in mst, em3d and 429.mcf, respectively. Average L2 MLP is increased by 26%, 0.3% and -1%, in mst, em3d and 429.mcf, respectively.

Keywords-chip multiprocessors; architectural simulation; data prefetching; memory system performance

I. INTRODUCTION

Data prefetching [1] is prevalent on multicores nowadays. Data prefetching speculatively issue memory requests ahead of demand requests, for data needed later by the main computation, in various forms such as hardware based stream prefetching, compiler inserted prefetch instructions, and helper thread based data prefetching. Data prefetching on chip multiprocessors can lead to increased stress on limited resources such as caches, memories and interconnects. Data prefetching, if not properly used, can degrade system performance by causing harmful effects such as cache pollution and waste of interconnect bandwidth. Therefore, accurate and fine grained measurement of the related runtime metrics is vitally important as the first step involved in reducing harmful prefetches and increasing memory level parallelism, and therefore improving the memory system performance of data prefetching on chip multiprocessors. However, the required measurement is prohibitively impossible on real machines without bringing nontrivial overhead and leading to inaccurate results.

In this paper, we use cycle accurate full-system simulation to study the memory system performance of our previously proposed helper thread based data prefetching with control of harmful prefetches on multicores - software-initiated inter-core LLC prepushing. For simplicity, a two-level cache hierarchy is assumed throughout this paper and only data prefetching for the shared L2 cache is considered. We modified the GEMS multiprocessor simulator to support trace-based measurement and offline analysis of two forms of parallelism inherent in the modern memory systems, e.g., memory level parallelism and DRAM bank level parallelism, and their relationship with software-initiated inter-core LLC prepushing.

Results show that, prepushing can achieve speedups of 1.628, 1.019 and 1.032 in mst, em3d and 429.mcf, respectively. Average L2 MLP is increased by 26%, 0.3% and -1%, in mst, em3d and 429.mcf, respectively. The speedup and increased average L2 MLP is trivial in em3d and 429.mcf, which is due to the small simulation scale that were adopted in our experiments. Furthermore, the availability of multi-bank DRAM controller can decrease the execution time by overlapping the servicing of increased concurrent off-chip L2 misses which are brought by software-initiated inter-core LLC prepushing.

The remaining of the paper is organized as follows. Section 2 introduces the general workflow of software-initiated inter-core LLC prepushing, and explains the MLP related calculations using an illustrative example. Section 3 elaborates on the experimental methodology that involved in the evaluation process. Section 4 presents the results. Section 5 shows the previous work. Section 6 concludes the paper.

II. BACKGROUND

A. Simplistic Scheme of Software-Initiated Inter-Core LLC Data Prepushing

In our previous proposed scheme of software-initiated inter-core LLC prepushing [2, 3], data prefetching tasks are offloaded from one CPU core to its underutilized neighboring core which shares the same L2 cache. The idea is to separate

Project (No. 61070029) Supported by the National Natural Science Foundation of China.

978-0-7695-4429-8/11 $26.00 © 2011 IEEE
DOI 10.1109/ISPAW.2011.56

and overlap the computation and memory accesses in target hotspot functions using the multicore, multithreaded hardware. A data prepushing thread is created on the neighboring core for speculatively pushing data from memory into the shared L2 cache ahead of the original thread, in the hope of reducing the overall execution time. The execution of the original thread and the data prepushing thread can therefore be pipelined and overlapped in time.

Fig. 1 shows the simplistic workflow of software-initiated inter-core LLC prepushing can be described as follows. (1) the data prepushing thread is spawned in main(), which is the entry point of the workload; (2) the data prepushing thread remains dormant until some caller of the target hotspot function has been invoked and code placed in the caller wakes up the data prepushing thread to let it start the prelude (specified by the parameter of lookahead); (3) the data prepushing thread enters a steady state of prepushing (specified by the parameters of blocksize and stride) until the execution of the program has passed some point(s) in the target hotspot function; (4) the data prepushing thread is suspended and waiting for the next turn of servicing hotspots; (5) after all the data prepushing work is done, the data prepushing thread is destroyed in main().

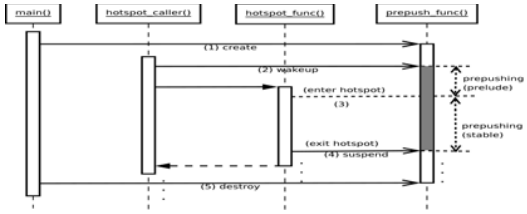

Figure 1. Simplistic Workflow of Software-Initiated Inter-Core LLC Prepushing

B. MLP Related Calculation: An Illustrative Example

A simple example can be given to illustrate how MLP can speedup execution and how the various MLP related calculation can be done. DRAM BLP related calculation can be done in a similar way as in L2 MLP calculation. As shown in Fig. 2, suppose there are three L2 misses A, B and C, and there is some overlapping in time when servicing these L2 misses. According to the definition of MLP, it is easily observed that MLP(c0,c1)=1, MLP(c1,c2)=2, MLP(c2,c3)=1, MLP(c3,c4)=2, MLP(c4,c5)=1.

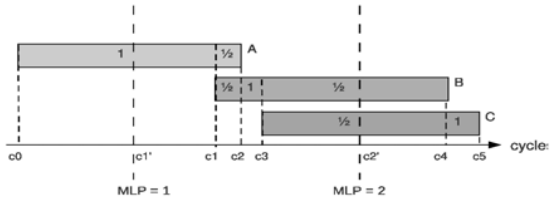

Figure 2. MLP and MLP-based cost Calculation

1) When MLP is not available (i.e., the three L2 misses are isolated misses that are serviced one by one without overlapping), the latencies or cycles spent servicing misses A, B and C are 350 cycles, 400 cycles and 380 cycles, respectively, therefore the accumulated total cycles spent servicing misses A, B and C is 1130 cycles.

2) When MLP is available (i.e., servicing multiple misses can be possibly overlapped and pipelined), the MLP-based costs of servicing misses A, B and C can be computed as in [4]: MLPCost(A) = Cycles(c0,c1)/MLP(c0,c1) + Cycles(c1,c2)/MLP(c1,c2) = 325 cycles, MLPCost(B) and MLPCost(C) can be computed similarly as 325 cycles and 225 cycles, respectively.

Then the total cycles spent servicing misses A, B and C can be computed in two ways: Cycles_MLP(A,B,C) = Cycles(c0,c1) + … + Cycles(c4,c5) = 780 cycles, or Cycles_MLP(A, B, C) = MLPCost(A) + MLPCost(B) + MLPCost(C) = 780 cycles.

Furthermore, the average MLP achieved in servicing misses A, B and C can be computed in two ways: avg_mlp(A,B,C) = Cycles_NoMLP(A,B,C) / Cycles_MLP(A, B, C) = 1.45 or avg_mlp(A,B,C) = Sum_of instantaneous_MLP_at_each_cycle / Cycles = 1.45.

III. Evaluation Methodology

A. Simulated Multicore System

Our simulated baseline multicore system design consists of two OoO cores, a two-level cache hierarchy and multi-banked DRAM physical memory. Both cache levels are lockup-free and store the state of outstanding requests using MSHRs. Each of the private first-level split caches are write-through with a coalescing store buffer. The shared second-level cache is write-back and maintains inclusion with respect to the first-level cache. A directory cache is stored on-chip.

Cycle-accurate detailed simulation of the DRAM memory controller system needs to be done to faithfully measure the BLP of incoming memory requests. This is done in GEMS by specifying a compatible cache coherence protocol. In our case, an implementation of the directory-based MESI protocol called MESI_CMP_filter_directory_m is used. Here, the postfix _m in the protocol name MESI_CMP_filter_directory_m indicates the detailed memory controller is used instead of a more simplistic but far less accurate fixed-latency model. A detailed multi-bank DDR 400 memory controller is simulated, to which the BLP measurement capability is added. Tab. I lists the detailed parameters of the simulated multicore system.

B. Simulator Configuration

GEMS [5] is selected as the base multicore simulator platform. By using Virtutech Simics [6] for trusted full-system functional simulation, GEMS provides mainly two components, i.e., Opal for modeling the microarchitecture of MIPS R10000-like five-stage OoO cores of SPARC v9 ISA, and Ruby for modeling cache coherence protocols, on-chip interconnects and DRAM memory controllers. Since we need to do cycle-

accurate miss profiling and MLP and BLP measurement, both Opal and Ruby are used. Simics 3.0.31 is used with the Cashew target, which implies UltraSPARC II cores and Aurora SPARC Linux 2.0. GEMS 2.1.1 is used in which the Opal module is augmented to support UltraSPARC II cores. The development host is an Intel Core 2 Duo E7400 machine running Ubuntu Linux 9.04. A pre-built GCC cross-compiler toolchain (sparc-linux-gnu 4.3.2) from Emdebian is used for ease. Software-initiated inter-core LLC prepushing has been manually implemented using the Pthreads library.

TABLE I. BASELINE HARDWARE CONFIGURATIONS

Processor	UltraSPARC II cores, frequency is 75 MHz, single chip
L1 I	64KB, 64B line-size, 4-way with LRU replacement; 4-wide fetch with 2-cycle request latency and 3-cycle response latency
Branch Predictor	YAGS with 20 PHT bits, 15 exception bits and 15 tag bits; minimum branch misprediction penalty is 1 cycle
Decode/Issue	4-wide; 8-entry reservation station; instruction window size is 32; ROB size is 64
Execution Units	4 INT ALUs, 2 INT DIVs; 2 BRANCHs; 4 FP ALUs, 2 FP MULTs, 2 FP DIV/SQRTs; 10 LOADs, 10 STOREs INT ALU: 1 cycle, MULT: 4 cycles, DIV: 20 cycles; BRANCH: 1 cycle; FP ADD/CMP/CVT: 2 cycles, MULT: 4 cycles, DIV: 12 cycles, SQRT: 24 cycles
L1 D	64KB, 64B line-size, 4-way with LRU replacement; 2-cycle request latency and 3-cycle response latency; 128 MSHRs
Unified L2	2MB, 64B line-size, 4-way with LRU replacement, inclusive; 4-cycle request latency, 6-cycle response latency and 6-cycle tag lookup latency; 128 MSHRs
Cache Coherence	Filter directory-based MESI CMP protocol; directory cache, 6-cycle latency
Interconnect	point-to-point topology, 14-cycle link latency
Memory	2GB, DDR 400; 8 banks x 2 ranks x 2 DIMMs x 1 channel; bus to memory frequency ratio of 10:1; cycle-accurate detailed simulation is enabled; maximum 12 outstanding requests per bank; bank busy time is 11 cycles; memory control latency is 36 cycles

C. Object Oriented Simulation Driver Scripting

To facilitate experimentation with GEMS with specific support of software based inter-core LLC prepushing, we developed a few object-oriented Python scripts to automate the simulation workflow which encompasses the following steps:

1) Configuration Injection. Parameter values (e.g., input set; prepush stride, step and blocksize; compiler flags) specified in the scripts should be pushed into related text files before benchmark compilation starts.

2) Benchmark Compilation. Related Makefiles should be invoked using proper parameters to compile benchmarks.

3) Simulation. There is another set of Python scripts to be invoked here for the real simulation with GEMS.

4) Statistics Extraction and Rendering. GEMS Statistics saved in the text files should be extracted and do necessary reporting and plotting work which is highly customizable.

An object-oriented class hierarchy was constructed to manipulate simulation options and results in a high level. Additionally, we add batch simulation capabilities in the scripting framework to make easy testing different combinations of configurations in a batch and compare them between each other.

D. Hotspot Oriented Two-Phase Simulation

Due to the specific needs of software-initiated inter-core LLC prepushing, The program state and execution flow is passed to the simulator by calling a few extra Simics magic call instructions as listed in Tab. II. ROI is the short name for region of interest. hotspot and main in the last column imply the corresponding Simics magic call is usually used in a hotspot function, the main function of a program, and the software-initiated inter-core prepushing thread function, respectively. When Simics or GEMS detects such magic call instructions, it will take appropriate actions to maintain simulation states and do predefined actions accordingly.

TABLE II. SIMICS MAGIC CALLS FOR HOTSPOT ORIENTED SIMULATION

Magic Call	Name	Insertion Location	Purpose
9001	Hotspot Begin	beginning of a hotspot function	notify the simulator the execution is entering a hotspot function
9002	Hotspot End	end of a hotspot function	if maximum number of executions reached, do warm checkpoint
9007	ROI Begin	after prepush thread is spawn	do cold checkpoint; begin profiled simulation
9008	ROI End	before prepush thread is destroyed	end detailed profiled simulation; do warm checkpoint

To ensure fast and accurate simulation on GEMS, we take a two-stage simulation approach: (1) Simics alone is used to fast forward the simulation of each workload until a predefined ROI Begin Simics magic call is executed and a cold checkpoint is taken; (2) The cold checkpoint is loaded and the profiled detailed simulation is performed using GEMS (hotspot and L2 miss profiling traces are dumped periodically during the period) until a predefined ROI End Simics magic call is executed; after that the simulation ends by calculating and saving the overall statistics to a predefined text file.

E. Trace Files for Detailed Measurement

A few trace file formats were designed to collect statistics for individual hotspot, retired instruction, L1 and L2 miss, and parallel DRAM request, and inter-core L2 load hit.

1) Hotspot Functions Trace File. The following metrics are collected for each predefined hotspot function: begin cycle, cycles spent, number of L2 misses, average L2 MLP and average DRAM BLP.

2) Retired Instructions Trace File. The following metrics are collected for each retired instruction: processor and thread ID, sequence number and type of instruction, dissembler representation, program counter, memory access address, access mode, cycle at which the instruction is fetched/retired/completed.

3) L2 Misses Trace File. The following metrics are collected for each L2 miss: whether the L2 miss has occurred within hotspot function or not, begin cycle, memory access address, DRAM DIMM/RANK/BANK ID, request type, program counter, instruction sequence number, access mode, whether it is demand miss or not, processor and thread ID, cycles spent, average L2 MLP and average DRAM BLP.

4) Parallel Memory Requests Trace File. The following metrics are collected for each bunch of parallel DRAM requests, the following metrics are collected: memory controller ID, current cycle, DRAM request distribution.

5) Inter-Core L2 Load Hits Trace File. The following metrics are collected for each pair of inter-core L2 load hits: memory access address, processor and thread ID and sequence number for two issuing instructions.

F. Measurement of L2 MLP and DRAM BLP

The information about the number of in-flight misses and the number of cycles a miss is waiting to get serviced can be tracked by the MSHRs found in various levels of the cache hierarchy. Each miss is allocated an MSHR entry before a request to service that miss is sent to the directory, and the MSHR entry is deallocated when a corresponding response is received from the directory.

Specifically, with respect to the allocation and deallocation of MSHRs (called transaction buffer entries (TBEs) in GEMS), part of L2 block state diagram of the MESI_CMP_filter_directory_m protocol that we used can be described as below: (1) An L2 data read miss occurs when an L1_GETS coherence message is received at the L2 cache and the requested cache block is not found in the L2 cache; an MSHR entry for the shared L2 cache will be allocated and a memory request will be sent to the directory and the cache block state in L2 will transition to a transient state ISS which signals waiting for the data from the directory. (2) When the requested data is returned from directory, the L2 cache will store and send the data back to the requester L1 cache; the previously allocated MSHR entry will be deallocated and the cache block state in L2 will be transition to a transient state ISS_MB which signals waiting to be unblocked.

Therefore, we approximate the number of cycles that an L2 miss stalls the processor by measuring the duration between the above allocation and deallocation action of the corresponding MSHR entry. MLP for in-flight L2 misses can be measured by how many L2 misses are being serviced concurrently when at least one miss is being serviced. Therefore, when doing MLP-based costs for parallel L2 misses, the stall cycles are divided equally among all concurrent in-flight L2 misses.

Similar to MLP, BLP can be achieved by servicing multiple memory requests simultaneously in different DRAM banks. BLP for in-flight memory requests and the corresponding L2 misses can be measured by how many DRAM banks are busy when at least one bank is busy servicing at least one memory request. For parallel requests, the cycles spent in servicing requests can be divided equally among all concurrent requests.

The information about the number of in-flight DRAM requests and the number of cycles a request is waiting to get serviced can be tracked by the response queue found in the DRAM memory controller. Each memory request is inserted into the response queue before the request gets to be serviced, and the memory request is removed from the response queue by the memory controller and sent back to the directory with the returned data or writeback acknowledgment after the request has completed servicing.

IV. RESULTS

All three benchmarks are cross-compiled with "-O3" and running on a simulated shared-cache dual-core machine. The prepush parameters for each benchmark are set empirically to mst: lookahead=20, blocksize=10; em3d: lookahead=10, blocksize=10; 429.mcf: lookahead=10, blocksize=10.

TABLE III. OVERALL PERFORMANCE

Mb[a]	*Pp*[b]	*Cycles*	*Speedup*	*Avg. L2 MLP*	*Avg. DRAM BLP*
mst (input: "1024 1"; hotspot: "HashLookup", max exec count: 100)					
N	N	236559382	1.0000	1.5137	1.2323
N	Y	145292785	1.6282	1.8695	1.4744
Y	N	218047768	1.0849	1.3327	1.3397
Y	Y	135442910	1.7466	1.7719	1.6485
em3d (input: "2000 100 75 1"; hotspot: "fill_from_fields", max exec count: 2)					
N	N	158697023	1.0000	0.3002	0.0854
N	Y	155626731	1.0197	0.3062	0.0873
Y	N	158529346	1.0011	0.2987	0.0855
Y	Y	155439732	1.0210	0.3039	0.0875
429.mcf (input: (test); hotspot: "refresh_potential", max exec count: 5)					
N	N	119294058	1.0000	0.6372	0.1034
N	Y	115515109	1.0327	0.6277	0.1020
Y	N	119054662	1.0020	0.6353	0.1038
Y	Y	115465217	1.0332	0.6258	0.1021

a. Availability of multi-bank DRAM controller

b. Availability of software-initiated Inter-Core LLC prepushing

Tab. III summarizes overall performance of the benchmarks, in which all the speedups are computed against the base case when there is only one bank in the DRAM controller and no software-initiated inter-core LLC prepushing.

Results show that, prepushing can achieve speedups of 1.628, 1.019 and 1.032 in mst, em3d and 429.mcf, respectively.

Average L2 MLP is increased by 26%, 0.3% and -1%, in mst, em3d and 429.mcf, respectively. The speedup and increased average L2 MLP of em3d or 429.mcf are trivial due to the small simulation scale adopted in our experiments, which should be improved in our further work.

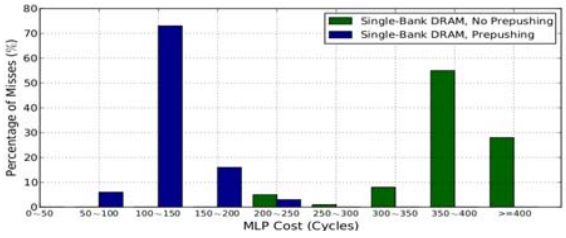

Figure 3. The Impact of Software-Initiated Inter-Core LLC Prepushing on MLP-based Cost Distribution of L2 Misses in mst

As depicted in Fig. 3, software-initiated inter-core LLC prepushing reduces the execution time by reducing the number of L2 cache misses occurred in the target hotspot function and increasing MLP when servicing multiple outstanding L2 cache misses. Isolated misses are more costly on performance than parallel misses. The non-uniformity of MLP can be observed in servicing L2 misses. The presence of software-initiated inter-core LLC prepushing can lower the MLP-based costs of a large portion of L2 misses, i.e., transform a large portion of isolated L2 misses into parallel L2 misses.

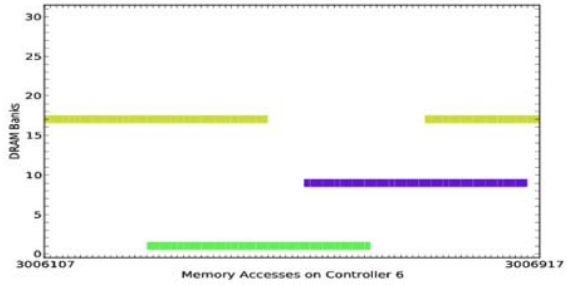

Figure 4. BLP Observed in Servicing of Memory Requests in mst

Furthermore, the availability of multi-bank DRAM controller further reduces the execution time by mapping the servicing of memory requests into multiple DRAM banks and increasing DRAM BLP when servicing multiple outstanding memory requests, as depicted in Fig. 4. This is especially useful for improving system performance when software-initiated inter-core data prepushing is enabled, where the increased concurrent L2 misses can inevitably congest the single-bank DRAM controller.

V. Related Work

MLP is studied often under such constraints as available bandwidth [7] and energy efficiency [8] of banked memories. Its impact on scheduling policies adopted in prefetching [9] (especially in the form of read miss clustering [10]) and cache partitioning/replacement [4, 11] schemes have been extensively investigated. Besides the availability of non-blocking caches and multi-banked DRAMs, achieving maximal benefits from MLP requires orchestration among different parts that support speculation and parallelism in the system.

In contrast to the implicit assumption in MLP studies that the DRAM latency of outstanding requests to memory will overlap, recent work on BLP shows that having a large number of outstanding requests queued in MSHRs [12] does not necessarily translates into overlapped latencies. If the requests in the DRAM controller's input queue are not to different banks, the amount of parallelism exploited in DRAM will be very low, reducing the effectiveness of aforementioned MLP improvement techniques. Consequently, BLP needs to be explicitly exploited by employing BLP-sensitive scheduling policies and mechanisms in various kinds of buffers, queues and memory controllers found in the memory hierarchy [13, 14, 9].

Memory system performance evaluation has been broadly used in various kinds of cache content management heuristics, either online [15] or offline [16], software [15] or hardware [16] or hybrid [17], prefetching [18] or cache replacement/partitioning [19]. It can be conducted either on real hardware [15, 20], or simulators [18]. The result is often stored in the form of trace files to be leveraged by offline training [21], or processed online to enable dynamic optimization [22].

VI. Conclusion

Experience on extending the Simics-GEMS simulator to evaluate the memory system performance of software-initiated inter-core LLC prepushing using memory-intensive benchmarks from Olden and CPU2006 is reported.

Results show that, prepushing can achieve speedups of 1.628, 1.019 and 1.032 in mst, em3d and 429.mcf, respectively. Average L2 MLP is increased by 26%, 0.3% and -1%, in mst, em3d and 429.mcf, respectively. The speedup and increased average L2 MLP of em3d or 429.mcf are trivial due to the small simulation scale adopted in our experiments, which should be improved in our further work. Furthermore, the availability of multi-bank DRAM controller can decrease the execution time by overlapping the servicing of increased concurrent off-chip L2 misses which are brought by software-initiated inter-core LLC prepushing.

Our proposed evaluation methodology can be an enabler for enhancing software-initiated inter-core LLC prepushing, e.g., adding MLP-awareness can improve its performance by scheduling data prepushes based on the MLP-based cost of each targeted cache misses.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under the contract No. 61070029.

REFERENCES

[1] S. Byna, Y. Chen, and X.-H. Sun, "Taxonomy of data prefetching for multicore processors," Journal of Computer Science and Technology, vol. 24, no. 3, pp. 405–417, 2009.

[2] Z. Gu, N. Zheng, Y. Zhang, C. Liu, J. Tang, and Y. Huang, "The stable conditions of a task-pair with helper-thread in cmp," in PDPTA, ser. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2009, Las Vegas, Nevada, USA, July 13-17, 2009, 2 Volumes, H. R. Arabnia, Ed. CSREA Press, 2009, pp. 125–130.

[3] J. Zhang, Z. Gu, N. Zheng, Y. Huang, M. Cai, S. Yang, and W. Zhou, "Performance evaluation of data-push thread on commercial cmp platform," in Networked Computing (INC), 2010 6th International Conference on, may 2010, pp. 1–6.

[4] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, "A case for mlp-aware cache replacement," in Proceedings of 33rd International Symposium on Computer Architecture (ISCA), 2006, pp. 167–178.

[5] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," SIGARCH Computer Architecture News, vol. 33, no. 4, pp. 92–99, 2005.

[6] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," IEEE Computer, vol. 35, no. 2, pp. 50–58, 2002.

[7] E. Ebrahimi, O. Mutlu, and Y. N. Patt, "Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems," in IEEE 15th International Symposium on High Performance Computer Architecture (HPCA), February 2009, pp. 7–17.

[8] O. Ozturk, G. Chen, M. Kandemir, and M. Karakov, "Cache miss clustering for banked memory systems," in Proceedings of IEEE/ACM International Conference on Computer-Aided Design, November 2006, pp. 244–250.

[9] C. J. Lee, V. Narasiman, O. Mutlu, and Y. N. Patt, "Improving memory bank-level parallelism in the presence of prefetching," in Proceedings of 42nd Annual International Symposium on Microarchitecture (MICRO), December 2009.

[10] V. S. Pai and S. Adve, "Code transformations to improve memory parallelism," in Proceedings of 32nd Annual International Symposium on Microarchitecture (MICRO), 1999, pp. 147–155.

[11] M. Moretó, F. J. Cazorla, A. Ramírez, and M. Valero, "Mlp-aware dynamic cache partitioning," in Proceedings of International Conference on High Performance Embedded Architectures and Compilers (HiPEAC), ser. Lecture Notes in Computer Science, P. Stenström, M. Dubois, M. Katevenis, R. Gupta, and T. Ungerer, Eds., vol. 4917. Springer, 2008, pp. 337–352.

[12] J. Tuck, L. Ceze, and J. Torrellas, "Scalable cache miss handling for high memory-level parallelism," in Proceedings of 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE Computer Society, 2006, pp. 409–422.

[13] O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems," in Proceedings of 35th International Symposium on Computer Architecture (ISCA). Beijing: ACM SIGARCH, June 2008.

[14] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt, "Prefetch-aware dram controllers," in Proceedings of 41st Annual International Symposium on Microarchitecture (MICRO). IEEE Computer Society, 2008, pp. 200–209.

[15] D. K. Tam, R. Azimi, L. B. Soares, and M. Stumm, "Rapidmrc: approximating l2 miss rate curves on commodity systems for online optimizations," in Proceedings of 14th international conference on Architectural support for programming languages and operating systems (ASPLOS). ACM, 2009, pp. 121–132.

[16] J. Kim, K. V. Palem, and W.-F. Wong, "A framework for data prefetching using off-line training of markovian predictors," in Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2002, pp. 340–347.

[17] J. Skeppstedt and M. Dubois, "Hybrid compiler/hardware prefetching for multiprocessors using low-overhead cache miss traps," in Proceedings of the International Conference on Parallel Processing (ICPP). IEEE Computer Society, 1997, pp. 298–305.

[18] S. Sharma, J. G. Beu, and T. M. Conte, "Spectral prefetcher: An effective mechanism for l2 cache prefetching," ACM Transactions on Architecture and Code Optimization (TACO), vol. 2, no. 4, pp. 423–450, 2005.

[19] J. Jeong and M. Dubois, "Cost-sensitive cache replacement algorithms," in Proceedings of 9th International Symposium on High-Performance Computer Architecture (HPCA). IEEE Computer Society, February 2003, p. 327.

[20] C. McCurdy and C. Fischer, "Using pin as a memory reference generator for multiprocessor simulation," SIGARCH Computer Architecture News, vol. 33, no. 5, pp. 39–44, 2005.

[21] S. Rubin, R. Bodík, and T. Chilimbi, "An efficient profile-analysis framework for data-layout optimizations," in Proceedings of 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL). ACM, 2002, pp. 140–153.

[22] D. Bruening, T. Garnett, and S. Amarasinghe, "An infrastructure for adaptive dynamic optimization," in Proceedings of International Symposium on Code Generation and Optimization (CGO), March 2003, pp. 265–275.