DRAM-Level Prefetching for Fully-Buffered DIMM: Design, Performance and Power Saving

Jiang Lin[1], Hongzhong Zheng[2], Zhichun Zhu[2], Zhao Zhang[1] and Howard David[3]

[1]Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011

[2]Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, IL 60607

[3]Digital Enterprise Group, Intel Corp., Hillsboro, OR 97124

Abstract

We have studied DRAM-level prefetching for the fully buffered DIMM (FB-DIMM) designed for multi-core processors. FB-DIMM has a unique two-level interconnect structure, with FB-DIMM channels *at the first-level connecting the memory controller and* Advanced Memory Buffers (AMBs)*; and DDR2 buses at the second-level connecting the AMBs with DRAM chips. We propose an AMB prefetching method that prefetches memory blocks from DRAM chips to AMBs. It utilizes the redundant bandwidth between the DRAM chips and AMBs but does not consume the crucial channel bandwidth. The proposed method fetches K memory blocks of L2 cache block sizes around the demanded block, where K is a small value ranging from two to eight. The method may also reduce the DRAM power consumption by merging some DRAM precharges and activations. Our cycle-accurate simulation shows that the average performance improvement is 16% for single-core and multi-core workloads constructed from memory-intensive SPEC2000 programs with software cache prefetching enabled; and no workload has negative speedup. We have found that the performance gain comes from the reduction of idle memory latency and the improvement of channel bandwidth utilization. We have also found that there is only a small overlap between the performance gains from the AMB prefetching and the software cache prefetching. The average of estimated power saving is 15%.*

1 Introduction

Multicore processors require high memory bandwidth because they multiply off-chip memory traffic, but it is challenging to meet the demand because off-chip memory bandwidth is limited by processor pin bandwidth [2]. The interconnect between the processor chip and the DRAM chips may have to be re-engineered to address the challenge. *Fully Buffered DIMM* (*FB-DIMM*) is a recent effort that uses narrow, high-speed memory channels with conventional DDR2 DRAM chips. It allows a processor to connect more memory channels and thus provides higher memory bandwidth. Meanwhile, it uses existing DDR2/DDR3 DRAM to avoid the high cost of adopting a new type of DRAM. From the viewpoint of memory subsystem design, FB-DIMM is an interesting subject for design optimization. It has a unique, two-level interconnect structure: The first level is a set of narrow, high-speed FB-DIMM memory channels and the second level is a larger set of conventional DDR2 buses[1]. The memory module of FB-DIMM looks like a conventional DRAM DIMM (Dual-Inline Memory Module) but with the addition of an *Advanced Memory Buffer* (*AMB*). The AMB provides the interface between an FB-DIMM channel and the DDR2 DRAM chips on the module. It converts the commands and data between the FB-DIMM channel format and the DDR2 format. The aggregate bandwidth of all DDR2 buses is greater than the FB-DIMM channel bandwidth when more than one FB-DIMM is connected to each channel. This redundancy of bandwidth can be utilized to improve the performance of a FB-DIMM memory subsystem.

We have studied DRAM-level prefetching for FB-DIMM that prefetches memory blocks from DRAM chips to AMB. We call it *AMB prefetching* hereafter. Prefetching at this level has a unique advantage: *It does not consume the crucial channel bandwidth to the processor chip*. By contrast, conventional prefetching to processor-level caches or memory controller may incur excessive memory traffic on the memory channels if the prefetching is not accurate, which will degrade the performance of multi-core processors. Our method works as follows: The AMB is extended to include a small SRAM cache of a few kilobytes; upon a demand memory request, the memory controller will fetch K memory blocks of the L2 cache block size around the demanded block with the demanded block included, where K is a small number. The DRAM interleaving is carefully revised to preserve this spatial locality at the DRAM level and meanwhile to allow memory access concurrency.

Many hardware and software methods of cache prefetching [23] have been proposed. They can prefetch data close to the CPUs, but most of them require redundant memory band-

*This work is supported in part by the National Science Foundation under grants CPA-0541408 and CPA-0541366 and the UIC WISER Fund.

[1]Future FB-DIMM will also support DDR3 bus and DRAM.

width which is less available in multi-core processors than in single-core processors. They may also cause cache pollution, which further increases memory traffic. Cached DRAM [9] adds a small SRAM cache inside DRAM chips as a prefetch buffer. It utilizes the huge bandwidth (hundreds of GB/s) inside the DRAM chips to explore the spatial locality: If one memory block is accessed, all other memory blocks in the same page will be prefetched in a single DRAM operation. The disadvantage is that it requires changes to the DRAM internal structure and external interface, but the price of DRAM is very sensitive to any of such changes. The AMB prefetching is somewhat between the cached DRAM and cache prefetching: It does not consume the crucial memory bandwidth to the processor or memory controller and does not require any change to the DRAM chips. Although the reduction of idle memory access latency is not as much as cache prefetching, it is still significant: In a regular setting of FB-DIMM, the idle memory access latency to the memory controller is reduced from 63ns to 33ns (see Section 4). Nevertheless, for memory intensive applications the idle memory access latency is a relatively small component when compared to the memory queueing delay, so the actual performance gain must be carefully evaluated by cycle-accurate simulation.

We have found that the AMB prefetching improves the performance of memory-intensive applications significantly. Our simulation results show that the AMB prefetching can improve the performance of SPEC-based memory-intensive workloads by 16% on average. Our analysis indicates that the performance gain comes from both the reduction of idle memory latency and the improvement of bandwidth utilization. The use of AMB cache reduces the accesses to the DRAM chips and therefore reduces bank conflicts, which are a major source of inefficient utilization of memory bandwidth. We have quantified the gains from the two sources, and found that the gain from better bandwidth utilization is more than that of idle latency reduction for multi-core processors.

We have made the following contributions in this study:

- We have thoroughly studied DRAM-level prefetching for FB-DIMM. We show that this type of prefetching is a different class when compared with cached DRAM or conventional cache prefetching, and prefetching at this level is surprisingly effective even with software cache prefetching enabled.
- We have proposed a region-based AMB prefetching method and have thoroughly evaluated its performance and analyzed the performance gain. We show that the performance gain is from both the idle latency reduction and the improved utilization of memory bandwidth.
- We show that DRAM-level prefetching works well with the existence of software cache prefetching, and their performance are very complementary.
- We have also evaluated the power saving by the proposed AMB prefetching and showed that it may save DRAM energy significantly. This energy saving is extra to the energy saving from the performance improvement.

The rest of this paper is organized as follows. Section 2 introduces the background of the FB-DIMM. Section 3 describes the proposed AMB prefetching. Section 4 describes the experimental environment and Section 5 presents the performance results and analyzes the performance gain of AMB prefetching. The related work is discussed in Section 6. Finally, Section 7 concludes this study.

2 Properties of Fully-Buffered DIMM

Fully-Buffered DIMM (FB-DIMM) is designed to scale with multi-core processors in both memory bandwidth and capacity. Today, a DDR2 memory channel using DDR2-800 chips can provide 6.4GB/s bandwidth. However, because of the stub bus structure of DDR2 and DDR3 channels, they can hardly maintain the signal integrity without reducing the number of memory devices (DRAM chips) and the wire length [24]. In other words, the maximum memory capacity per channel may have to drop with the increase of bandwidth. Furthermore, DDR2 or DDR3 channels use a large number of pins (240 pins for DDR2 DIMM used in desktop computers), which limits the number of channels that can be put on a motherboard.

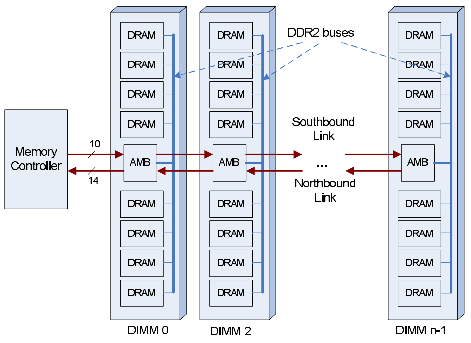

Figure 1: The structure of fully-buffered DIMM with one channel, n DIMMs and eight DRAM chips per DIMM. The memory controller is able to connect at least six channels, and each channel may connect up to eight DIMMs.

Figure 1 shows the structure of FB-DIMM with one channel connecting n DIMMs. It has a two-level interconnect structure, the FB-DIMM channel and the DDR2 buses on the DIMMs[2]. The AMB (Advanced Memory Buffer) is a key component in this interconnect structure. The memory controller links to those AMBs through a narrow but high frequency point-to-point bus, forming a daisy chain. Figure 1 shows only one channel connected to the memory controller; in real systems, multiple channels can be connected to a single controller. The DRAM chips on a DIMM are connected to the DIMM's AMB; they are not directly connected to the channel bus. The narrow bus runs at a much higher frequency than the DDR2/DDR3 bus, significantly reducing the number of pins needed per memory channel. The number of pins per channel is 69 with a default configuration. In addition, the point-to-point, daisy-chain connection allows a FB-DIMM channel to

[2]Unlike in conventional DDR2 memory, here one bus only connects a single set of DRAM chips.

support more DIMMs at the cost of increased latency. More channels and more DIMMs per channel mean the FB-DIMM technology can support higher memory capacity. Meanwhile, the use of AMB leaves the DRAM chips unchanged.

The FB-DIMM channel interconnect has two unidirectional links, southbound link and northbound link, which operate independently. The southbound link has ten logical signals and may carry memory commands and data to be written; and the northbound link typically has fourteen logical signals and carries the read data returned from the DIMMs. Each logical signal is carried by a pair of wires using differential signaling. The memory controller schedules the commands and data transfers on both links. During each memory cycle, the southbound link can transfer three commands or one command and 16-byte write data; and the northbound link can transfer 32-byte read data. The maximum bandwidth of the northbound link matches that of one DDR2 channel. In the future, the FB-DIMM will support DIMMs using DDR3 channel. A point worth noting is that the overall bandwidth of a FB-DIMM channel is higher than that of a DDR2 channel because the write bandwidth is extra.

The AMB is a small logic component attached to each DIMM and sits between the memory controller and DRAM chips. It receives commands and data from the FB-DIMM channel; and then determines whether the commands and data are for its memory devices or not. If yes, the AMB will translate the commands and data from the FB-DIMM channel format to the internal DDR2/DDR3 format; otherwise, it will forward the commands and data to the next AMB or the memory controller along the FB-DIMM channel. An important feature of the FB-DIMM is that it has variable read latency (VRL). The minimum latency of accessing a given DIMM depends on its logic distance from the memory controller. In other words, a DIMM close to the memory controller may provide return data in a shorter latency than a remote DIMM. The FB-DIMM can also be configured to not supporting the VRL feature. In that case, every DIMM has a fix minimum read latency, which is the latency of the farthest DIMM.

3 AMB Prefetching — DRAM-Level Prefetching for FB-DIMM

In this section, we first discuss the working principles of AMB prefetching and then describe the design of a region-based AMB prefetching.

3.1 Working Principles of AMB Prefetching

The AMB prefetching prefetches memory blocks from DRAM chips to a small cache attached to the Advanced Memory Buffer. In FB-DIMM, the bandwidth of northbound link (for read) matches the bandwidth between the AMB and memory chips of a single DIMM, and the bandwidth of southbound link (for write) is half of that of northbound link. In a FB-DIMM channel with multiple DIMMs, the aggregate DIMM-level bandwidth is significantly higher than the channel bandwidth. For example, if a FB-DIMM channel has four DIMMs with DDR2 chips of 800MT/s (Mega Transfers/second) speed, the aggregate bandwidth between AMBs and memory chips is $6.4GB/s \times 4 = 25.6GB/s$. In comparison, the northbound bandwidth is 6.4GB/s; and the southbound bandwidth is 3.2GB/s, which means the total bandwidth of FB-DIMM channel is 9.6GB/s.

The benefit of AMB prefetching is twofold: AMB prefetching shortens the idle memory access latency for AMB-cache hit and improves the utilization of the FB-DIMM channels. The idle memory access latency is the time to serve a memory request when the memory subsystem is idle; in other words, there is no queueing delay. A DRAM access typically involves up to three DRAM operations: precharge, activation (row access), and data access (column access). If a request hits in the AMB cache, there is no need for those DRAM operations, and thus the latency is reduced. In a typical setting, which is used as default in our experiment, the minimum idle latency without the AMB cache is the 63ns, and is reduced to 33ns with the AMB cache. The utilization of channel is also improved because the AMB cache reduces the access frequency to the DRAM and therefore reduces the DRAM bank conflicts, which are a major source of inefficiency in the utilization of memory bandwidth. If an access hits in the AMB cache, the data can be transferred immediately. By contrast, in the original FB-DIMM, every access will fall into the DRAM chips. If that requires DRAM precharge and/or activation, the FB-DIMM channel may have to idle.

The working principle of the AMB prefetching is somewhat close to that of cached DRAM [9]: It utilizes the spatial locality in the memory access stream. It is known that the accesses to the memory subsystems, being filtered by caches, have little temporal locality but good spatial locality [26]. It is also known that a small cache may well utilize this locality [10, 7, 12, 27]. Therefore, the AMB cache does not have to be large. We find that an AMB cache of a few KBs is sufficient.

3.2 The Design of A Region-Based AMB Prefetching

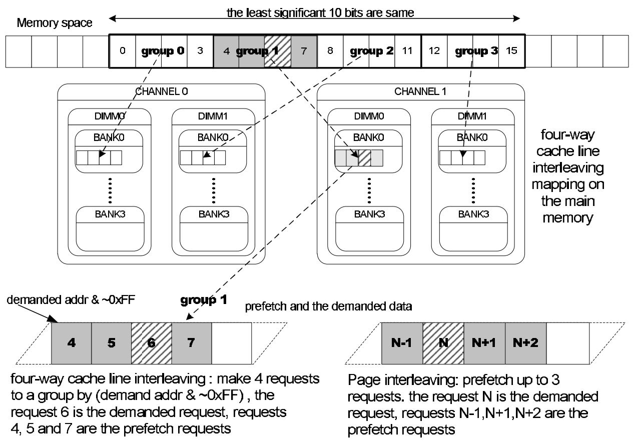
Memory space
the least significant 10 bits are same
0
group 0
3
4
group 1
7
8
group 2
11
12
group 3
15
CHANNEL 0
CHANNEL 1
DIMM0
DIMM1
DIMM0
DIMM1
BANK0
BANK3
four-way cache line interleaving mapping on the main memory
demanded addr & ~0xFF
group 1
prefetch and the demanded data
4
5
6
7
N-1
N
N+1
N+2
four-way cache line interleaving : make 4 requests to a group by (demand addr & ~0xFF) , the request 6 is the demanded request, requests 4, 5 and 7 are the prefetch requests
Page interleaving: prefetch up to 3 requests. the request N is the demanded request, requests N-1,N+1,N+2 are the prefetch requests

Figure 2: The data layout of four-way cacheline interleaving and FB-DIMM AMB prefetching.

For simplicity, hereafter *AMB prefetching* refers to this particular region-based prefetching method that we designed for FB-DIMM. The full design of AMB prefetching includes the following changes: (1) the addition of a small prefetch buffer, called *AMB cache*, to each AMB; (2) a tag structure in the memory controller to trace prefetched data; (3) an extension to the AMB to issue multiple pipelined column access commands, which are normal DRAM column access operations, to fetch data from DRAM to the AMB-cache; and finally (4) an extension to the AMB to return data directly from its AMB-cache.

The region-based prefetching is a particular way to predict the prefetching addresses [13]. Our AMB prefetching may fetch up to K cachelines on a single memory read request, where K is a small number (ranging from two to eight in our experiments). Basically, the physical memory access space is divided into regions of the same size of K cachelines. For each demand read request, i.e. memory request generated by demand cache miss from the processor, the memory controller will send a special column access command to the AMB. The AMB will first issue a column access command to DRAM for the demanded data, and then send $k - 1$ column access commands to DRAM for the prefetching data.

The AMB prefetching requires certain DRAM interleaving schemes. A DRAM interleaving scheme determines how memory addresses are laid out onto channels, DIMMs and DRAM banks. Each DIMM may have multiple ranks. Each rank consists of multiple DRAM chips, which are ganged together to form a 64-bit logic data path. Only one of the ranks can be accessed at a time. Each DRAM chip usually have four or eight internal DRAM banks, and therefore each DIMM has four or eight *logic* DRAM banks formed by physical DRAM banks distributed onto the chips belonging to the same rank. All physical DRAM banks in a logic DRAM bank are precharged, activated or column accessed at the same time. Hereafter *DRAM bank* refers to a logic DRAM bank if not mentioned otherwise. Two types of DRAM interleaving are commonly used: cacheline interleaving and page interleaving. With cacheline interleaving, consecutive cachelines in the physical memory address space are laid out onto the channels, DIMMs, ranks and DRAM banks sequentially with wraparound. It is good for exploiting memory access currency among cachelines. Page interleaving is similar except that it uses page as the interleaving granularity; here the page size is the page size of a physical DRAM chip multiplied by the number of DRAM chips per rank. Page interleaving is good for exploiting spatial locality: If two cachelines being accessed are mapped to one DRAM bank, only one precharge and activation is needed.

We use a multi-cacheline interleaving scheme for AMB prefetching to exploit both spatial locality and memory access concurrency. In this scheme, the interleaving granularity is K cachelines. When one cacheline (precisely a memory block of cacheline size) is fetched, the other $K - 1$ cachelines in the same region, which are mapped to the same logic DRAM bank, are transferred to the AMB cache without extra DRAM precharge and activation. A conventional cacheline interleaving would map consecutive cachelines onto different DRAM banks or DIMMs, and therefore incur multiple DRAM precharges and activations. Figure 2 shows an example of the four-way cacheline interleaving scheme in a two-channel system with a single rank per DIMM, two DIMMs per rank and four internal banks per DIMM. Here four consecutive cachelines make a group that is mapped to the same DRAM page; and the continuous groups are mapped to different channels and banks in a round-robin way. Assume that the AMB prefetching is applied and the demanded request is on block six, the corresponding group which includes blocks four, five and seven will be prefetched. The AMB prefetching may also be used with page interleaving, also shown in Figure 2. If the demanded request is on block N, blocks N-1, N+1 and N+2 will be prefetched into the prefetch buffer as long as they are in the same page. Note that the open page mode should be used with page interleaving, but the close page mode should be used with cacheline interleaving or multi-cacheline interleaving. In practice, the close page mode is widely used because it has lower implementation complexity.

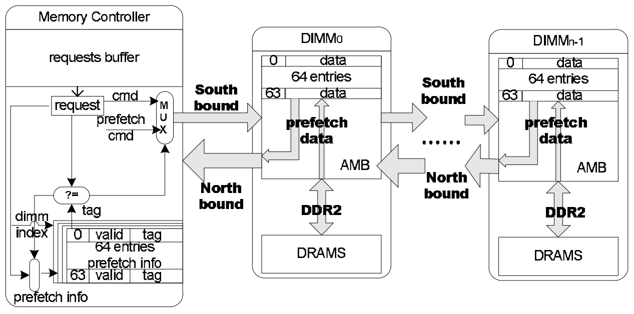

Figure 3: Structure of AMB prefetching for FB-DIMM system.

Figure 3 shows the overall structure of AMB prefetching. AMB is a relatively large chip and it is feasible to add a small SRAM cache as a prefetch buffer. The memory controller holds the tag part of the cache and the AMBs hold the data part. In our default configuration, each AMB cache holds 64 cachelines of 64 bytes each and 4KB in total. The tags and status bits of all AMB cache contents are stored in a prefetch information table at the memory controller. When a new read request arrives, the memory controller checks to see if it hits in the corresponding AMB cache (the data has been prefetched). If it hits, the memory controller sends a special fetch command to the AMB; and the AMB will return the data from its cache. If it misses, the memory controller will send a command to the AMB to fetch a group of K cachelines. The demanded cacheline will be fetched first and forwarded to the controller; and the other prefetched cachelines will be stored in the AMB-cache. The memory controller updates the prefetch information table accordingly. This design does not require any change to the physical connection between the memory controller and the AMB, but only some changes to the command protocol. In the original FB-DIMM, the memory controller sends commands to the AMB in packets; this design will add a few new encodings to the command packet and logics at the AMB to respond to those new commands. There is no change to the

DRAM chips. This design does *not* change the burst length of the DDR2 bus on each DIMM: The AMB simply issues multiple, column accesses to the DRAM chips; and the multiple data transfers on the DDR2 bus are fully pipelined. The AMB-cache replacement policy is FIFO; LRU is not suitable for AMB cache because a hit block may be cached in the processor and will not be accessed soon. If in a DRAM bank the currently active DRAM page gets replaced, its cachelines may stay in the AMB cache until they are replaced.

4 Experimental Methodology

4.1 Simulation Environment

We use M5 [1] as the base simulator and extend the memory part for simulating multi-channel FB-DIMM with DDR2 memory devices. Software prefetch is supported by M5 and used in our experiments. The memory simulator uses a cycle-driven framework that is able to model the memory access reordering. It also simulates the details of FB-DIMM northbound and southbound links, isolated command and data buses inside FB-DIMM, and shared command and data bus for DDR2 channel. The simulated memory controller uses the hit-first policy [18] to reorder pending memory requests, under which row buffer hits are scheduled prior to row buffer misses. In addition, read requests are scheduled before write requests unless the number of outstanding write requests is above a certain threshold. Table 1 shows the major parameters of the pipeline and the memory subsystem; and Table 2 further shows the major parameters of DRAM operations.

Parameters	Values
# of proc. cores	1/2/4/8
Proc. speed	4 GHz
Pipeline	8-issue, 21-stage
FUs	6 IntALU, 2 IntMult, 4 FPALU, 2 FPMult
Issue queue	64 entries
ROB	196 entries
Phy. reg.	228 Int, 228 FP
LQ/SQ	32 LQ entries, 32 SQ entries
Branch predictor	Hybrid, 8k global + 2K local, 16-entry RAS, 4K-entry and 4-way BTB
L1 caches (per core)	64KB Inst/64KB Data, 2-way, 64B line, hit latency: 1 cycle Inst/3-cycle Data
L2 cache (shared)	4MB, 4-way, 64B line, 15-cycle hit latency
MSHR entries	Inst:8, Data:32, L2:64
Memory	2/4/8 logic channel, 2 physical channels/logic channel, 2/4/8 DIMMs/physical channel, 4 banks/DIMM
Channel bandwidth	667MT/s (Mega Transfers/second), DDR2/FB-DIMM-DDR2
Memory buffer	64 entries
Memory controller overhead	12ns

Table 1: Simulator parameters.

4.2 Workload Construction

In our experiments, each processor core is single-threaded and runs a distinct application. From the SPEC2000 benchmark suite [22], we select twelve memory-intensive programs: *wupwise, swim, mgrid, applu, vpr, equake, facerec, lucas,*

Para.	Value	Meaning (delay of)
tRP	15ns	PRE to ACT to same bank
tRCD	15ns	ACT cmd to RD cmd to the same bank
tCL	15ns	RD cmd to RD DATA
tRC	54ns	ACT cmd to ACT cmd to the same bank
tRRD	9ns	ACT cmd to ACT cmd or PRE cmd to PRE cmd to different banks
tRPD	9ns	RD cmd to PRE cmd
tWTR	9ns	WR DATA end to RD cmd
tRAS	39ns	ACT cmd to PRE cmd for reads
tWL	12ns	WR cmd to WR data bus cycles
tWPD	36ns	WR cmd to PRE cmd

Table 2: DRAM parameters.

fma3d, parser, gap and *vortex*. We construct the multiprogramming workloads randomly from these selected applications, which are shown in Table 3. Each workload is named with a format of *number of applications–X (workload index)*. For example, the workload 2C-1 consists of two applications, *wupwise* and *swim*. We exclude two memory intensive programs, *art* and *mcf*. Program *art* has very low miss rate with 4MB cache and very high miss rate with 2MB cache [8], therefore its cache miss rate will become unpredictable in our settting. Program *mcf* has very low IPC and its relative performance gain or loss may bias the overall performance gain or loss.

Group	Name	Benchmarks
2-core	2C-1	wupwise, swim
	2C-2	mgrid, applu
	2C-3	vpr, equake
	2C-4	facerec, lucas
	2C-5	fma3d, parser
	2C-6	gap, vortex
4-core	4C-1	wupwise, swim, mgrid, applu
	4C-2	vpr, equake, facerec, lucas
	4C-3	fma3d, parser, gap, vortex
	4C-4	wupwise, mgrid, vpr, facerec
	4C-5	fma3d, gap, swim, applu
	4C-6	equake, lucas, parser, vortex
8-core	8C-1	wupwise, swim, mgrid, applu, vpr, equake, facerec, lucas
	8C-2	wupwise, swim, mgrid, applu, fma3d, parser, gap, vortex
	8C-3	vpr, equake, facerec, lucas, fma3d, parser, gap, vortex

Table 3: The makeup of workloads.

It is infeasible to run SPEC applications to the completion in simulation, especially for multi-core workloads. We use the simulation points suggested by SimPoint 3.0 [19] to control the simulation time. The simulation stops when one processor core commits 100 million instructions, the length of a simulation point. We use the SMTspeedup [21] as the metrics to compare the performance of multi-core execution. It is calculated as SMT speedup $= \sum_{i=1}^{n}(\text{IPC}_{\text{cmp}}[i]/\text{IPC}_{\text{single}}[i])$, where n is the number of cores, $\text{IPC}_{\text{cmp}}[i]$ is the IPC of the program running on the ith core and $\text{IPC}_{\text{single}}[i]$ is the IPC of the same program running on the single-core environment.

5 Performance Evaluation and Analysis

We first compare the performance of FB-DIMM and DDR2 briefly as FB-DIMM is a new design. We then present the performance of AMB prefetching, compare several of its variants, and analyze the sources of performance gain, and then evaluate the power saving from AMB prefetching.

In the default setting, the FB-DIMM memory subsystem consists of four memory channels (DDR2 or FB-DIMM) with data rate of 667MT/s per channel, four DIMMs per channel and four banks per DIMM, with two memory channels ganged together to form a logic channel. For the DDR2 memory and FB-DIMM without prefetching, the cacheline interleaving with the close page mode is used. For FB-DIMM with AMB prefetching, multi-cacheline interleaving with the close page mode is used. Software prefetching to the processor cache is enabled in the default setting. VRL is not used with FB-DIMM. We did experiment with VRL and found that the performance improvement from the AMB prefetching is very similar to that without VRL; the result is not presented due to the space limit.

5.1 Comparison of FB-DIMM and DDR2 DRAM

We first compare the performance of FB-DIMM with DDR2. (A recent study [6] has evaluated FB-DIMM in detail.) Figure 4 shows the SMT speedup of the workloads with DDR2 and FB-DIMM, using single-threaded execution with DDR2 as the reference points. Thus, the SMT speedup for single-threaded execution with DDR2 is 1.0. Both of the DDR2 and FB-DIMM configurations have four channels with comparable bandwidth; the FB-DIMM bandwidth is slightly higher because the FB-DIMM channel has dedicate data path for memory writes. Overall, the FB-DIMM system performs comparably or slightly worse than the DDR2 system on single-core and dual-core processors. The average loss is 1.5% and 0.6%, respectively. However, the FB-DIMM performs better on four-core and eight-core processors; the average gain is 1.1% and 6.0%, respectively. Those observations are not surprising: FB-DIMM has longer idle latency than DDR2 but has higher bandwidth and may utilize its bandwidth more efficiently.

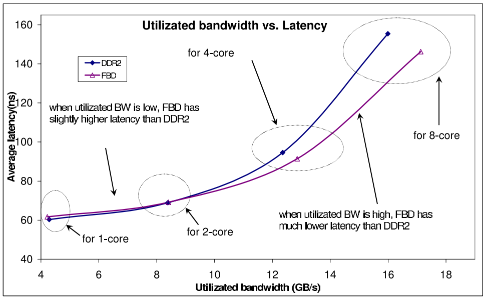

Figure 5: The average of utilized bandwidth vs. the average latency for DDR2 and FB-DIMM.

Memory latency and bandwidth utilization. Figure 5 shows the average utilized bandwidth (the X axis) and the average latency (the Y axis) observed in the simulation. We present the utilized bandwidth (or bandwidth usage) instead of bandwidth utilization because the FB-DIMM and the DDR2 have different total peak bandwidth. The results show that the average latency is shorter on the FB-DIMM than on the DDR2 for multi-core configurations. For example, the average of utilized bandwidth is the highest with the eight-core workloads, with 17.1GB/s for FB-DIMM and 16.0GB/s for DDR2. The average latency of FB-DIMM is only 146ns, compared with the 155ns average latency of DDR2. As expected, the single-core workloads utilize much less bandwidth, with only 4.2GB/s on average for both FB-DIMM and DDR2 configurations. The average latency is 62ns and 60ns, respectively. The observed average latency of FB-DIMM is slightly higher than that of DDR2, since the idle latency of FB-DIMM is higher and single-core workloads have relatively low memory-level parallelism. For workloads whose utilized bandwidth is high, e.g. 2C-1, 2C-2, 4C-1, 4C-3, 4C-5 and all three 8-core workloads, the FB-DIMM system achieves better performance than the DDR2 system. All three 8-core workloads utilize more than 14GB/s bandwidth, and their performance on FB-DIMM is 6.0% better.

Performance of FB-DIMM with varying data rates and numbers of channel. Figure 6 presents the overall performance by varying the data channel rate and the number of memory channels (hereafter channel refers to logic channel), which also changes the total memory bandwidth. As expected, the overall system performance improves as the memory bandwidth increases. The above mentioned observations still apply: On average, the DDR2 performs slightly better for single-core and dual-core workloads; and the FB-DIMM performs better for four-core and eight-core workloads. The figure also quantifies the performance gain of increasing the data rate and the number of channels. For example, the gain by increasing the data rate from 533MT/s to 667MT/s is 12.7% on the FB-DIMM (FBD) for single-core workloads, and jumps to 20.5% for four-core workloads. The gain by increasing the number of channels from one to two is 8.8%, and another 5.1% from two to four, for single-core workloads. The improvement for eight-core workloads, however, is 75.1% from one channel to two channels and 49.0% from two to four.

Worth noting is that FB-DIMM scales much better than the DDR2/DDR3 memories with the increase of data rate and number of channels. As the data rate increases, it is increasingly challenging to maintain the signal integrity of the DDR2/DDR3 bus. On the other hand, FB-DIMM can support up to eight DIMMs at high data rate because of the point-to-point communication between AMBs; and a FB-DIMM channel requires much less pins (about 70 pins vs. 240 of DDR2), making it possible (or cheaper) to use more channels.

5.2 Overall Performance of AMB Prefetching and Performance Analysis

Overall performance. Figure 7 shows the overall performance improvements by using AMB prefetching. The default configuration is used: two FB-DIMM logical channels, four-cacheline interleaving, 64 AMB cache blocks of 64 bytes each, fully associated AMB cache, and software cache prefetching turned on. The reference points used in the SMT speedup are the individual programs' execution times on *single-core* processor with two-channel *DDR2* memory. The overall improve-

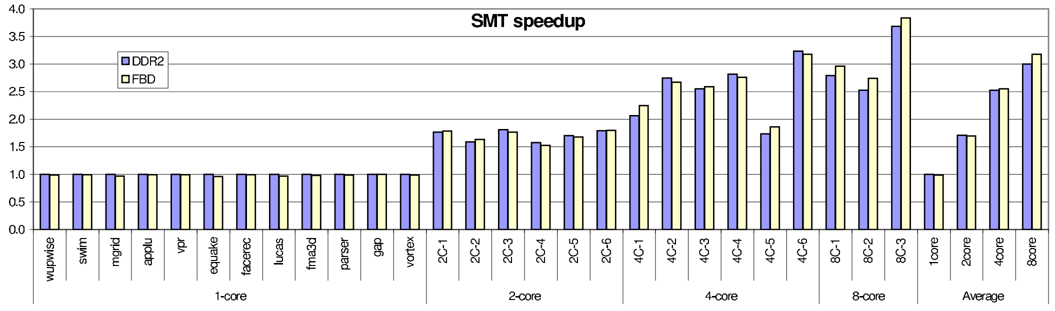

Figure 4: SMT speedup of 1-, 2-, 4- and 8-core execution with memory systems of DDR2 and FB-DIMM.

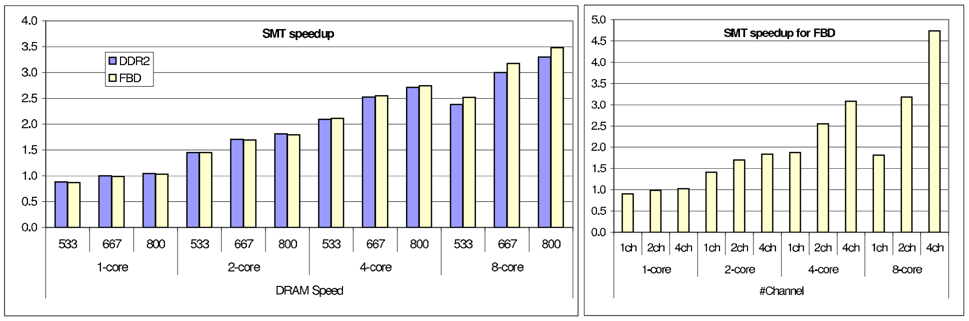

Figure 6: Bandwidth impact on performance.

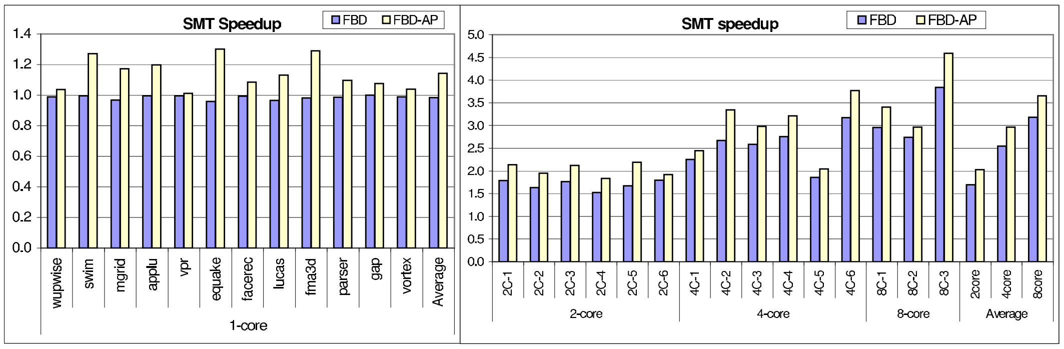

Figure 7: Performance of AMB-Prefetching (FBD:/FBD-AP: FB-DIMM without/with AMB prefetching).

ment is significant: The average improvement, measured by the average SMT speedup of FB-DIMM with AMB prefetching over the average SMT speedup of FB-DIMM without AMB prefetching, is 16.0%, 19.4%, 16.3% and 15.0% for single-, dual-, four- and eight-core workloads, respectively. The maximum improvement on dual-, four- and eight-cores are 30.7%, 25.1% and 19.7%, respectively. Considering the relatively simple changes by AMB prefetching to the whole system, this degree of improvement is significant. Worth noting is that the AMB prefetching also improves the performance for the single-core workloads; and unlike the conventional FB-DIMM, it is always better than DDR2 [3].

Prefetch coverage and efficiency. Figure 8 shows the prefetch coverage and efficiency (accuracy). The prefetch coverage is defined as $coverage = \#prefetch\_hit/\#read$. Prefetch efficiency is defined as $efficiency = \#prefetch\_hit/\#prefetch$. Our data show that the AMB prefetching has high coverage. The theoretical upper bound of prefetch coverage for four-cacheline interleaving is 75%. The AMB prefetching achieves around 50% prefetch coverage for those configurations with four-cacheline interleaving. As one might expect, increasing the prefetch buffer size and associativity can improve both the prefetch coverage and efficiency. However, increasing the prefetching ratio K (and the interleaving size) has negative impact on the prefetch efficiency but positive impact on the prefetch coverage.

Quantifying the sources of performance gain. There are two major sources of the performance gain from the AMB prefetching: The reduction of idle memory latency and the improvement of channel bandwidth utilization. For each DRAM access, the idle latency in the memory subsystem in the default configuration is 63ns: 12ns for memory controller over-

[3]The figure cannot show the direct comparison between FB-DIMM with DDR2 except for single-core workloads.

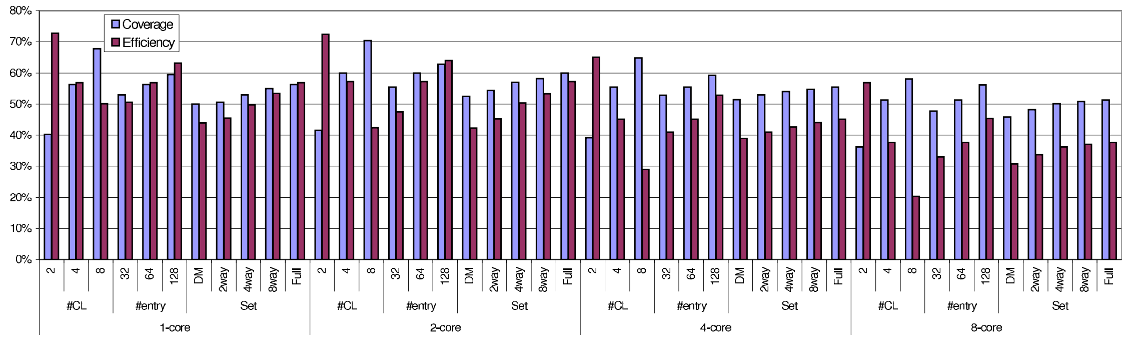

Figure 8: AMB-prefetch coverage and efficiency for varying number of cachelines in a region (#CL), number of blocks in the AMB cache (#entry) and set associativity (Set). The default setting is with #CL=2, #entry=64 and full set associativity. The bars with #CL=4, #entry=64 and Set=Full are identical and are shown for completeness.

head, 3ns for channel command delay, 15ns for activation (row access), 15ns for column access, 6ns for channel data transfer delay, and 3ns $\times$ 4 = 12ns for the extra delay at the four AMBs. For each AMB hit, the idle latency is 33ns, i.e. 30ns less for eliminating activation and column access. Worth noting is that the CPU sees longer latency due to on-chip cache processing. If two accesses fall onto the same DRAM bank but to different pages, they cause bank conflicts and the two DRAM activations must be separated by 54ns. If there are no enough memory requests to fill this gap (nine requests needed in the default configuration), the channel will have to idle and the channel bandwidth utilization is reduced. Bank conflicts appear in single-core setting because many applications have multiple access streams; and the degree of conflict increases with the number of cores.

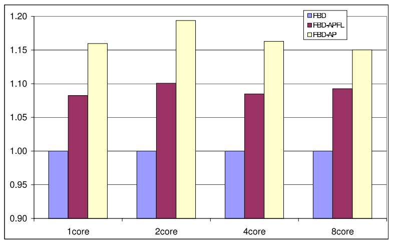

Figure 9: Decomposition performance gain.

We want to see how much performance is from the latency reduction and how much is from the bandwidth utilization improvement. To quantify the effects, we design another set of experiments in which each AMB-cache hit has the same idle latency as an AMB-cache miss; however, the DRAM bank is not activated. In other words, the AMB cache does not reduce the idle latency but only bank conflicts. We call it *AMB Prefetching with Full Latency.* Figure 9 compares its average performance (shown as FBD-APFL) with FB-DIMM (FBD) and FB-DIMM with AMB prefetching (FBD-AP). The difference between FBD and FBD-APFL indicates the performance gain for better bandwidth utilization, and the difference between FBD-APFL and FBD-AP indicates the gain for idle latency reduction. The gains are 8.2%, 10.1%, 8.5% and 9.2% from better bandwidth utilization, and 7.1%, 8.5%, 7.2% and 5.3% for idle latency reduction, for single-, dual-, four-, and eight-core workloads, respectively. The gains from the two sources are comparable. For eight cores, the gain from better bandwidth utilization is higher than that of latency reduction, which is expected because the processor is now more affected by bandwidth than latency.

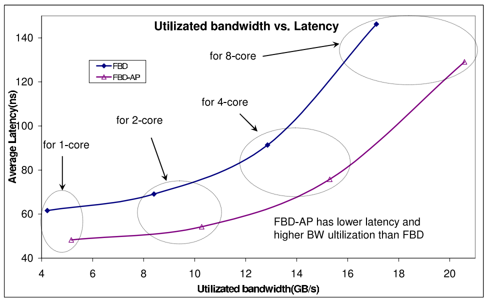

Figure 10: The average of utilized bandwidth vs. average latency for FB-DIMM with and without AMB prefetching.

AMB-prefetching and FB-DIMM channel bandwidth utilization. Figure 10 compares the average of utilized bandwidth of FB-DIMM with (FBD-AP) and without (FBD) AMB prefetching. For every workload, the utilized bandwidth of FBD-AP is significantly higher than FBD, and the latency is significantly shorter.

5.3 Sensitivity of AMB-Prefetching Performance to Configuration Variants

The default configuration of AMB prefetching uses fully associative and 64 cache lines (4KB) for prefetch buffer with four-cacheline interleaving. Figures 11 shows how the performance changes when the set associativity, prefetch buffer size and the granularity of cacheline interleaving vary. The number of prefetched cachelines is the same as the granularity of cacheline interleaving. All performance data are normalized to that of the default setting. The first three bars of each group has the interleaving size (#CL) increasing from two to eight. As the results show, single- and dual-core workloads get higher performance with more cachelines prefetched per demand access. However, for four- and eight-core workloads, four-cacheline interleaving yields the best performance. A larger prefetch size means a possibly higher prefetch hit rate, but also longer waiting time for on-demand accesses.

The middle three bars of each group show the performance

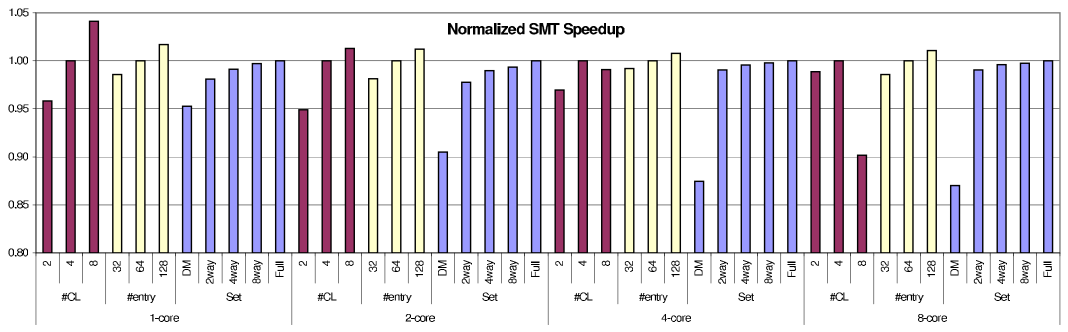

Figure 11: Sensitivity analysis.

with three prefetch sizes, 32, 64 and 128 cachelines, with four-cache interleaving and full associativity. The performance results are close. This indicates that 32 (2KB) or 64 (4KB) are large enough for those workloads.

As one might expect, the fully-associative buffer has the highest performance. However, the two-way associativity can achieve above 98% performance of that of the full associativity. The direct mapping does not work well, though, and only achieves 95.3%,90.5%,87.4% and 87.0% performnace of that of the full associativity for single-, two-, four- and eight-core workloads. In practice, we may use the two-way and four-way set associativity for power saving at the memory controller if that is a concern. Worth noting is that the set associativity only affects the power consumption of the tag structure stored at the memory controller, but not that of the AMB cache structure. From the cache design perspective, the AMB cache uses sequential access for the tag and data parts.

5.4 AMB Prefetching and Software Cache Prefetching

An important question is how the AMB prefetching works with existing cache prefetching methods. Since there are a large volume of hardware and software prefetching methods, and the evaluation of those methods is complex by itself, we choose a relatively simple and reliable method. In the default configuration, we have turned on the software prefetching: This is done by setting an option in the m5 simulator to execute software prefetching instructions in the binary code. The SPEC binary code is pre-built on DEC Alpha computer [25]; the compiler has generated software prefetching instructions. Software cache prefetching is enabled in the default setting. In this part of experiment, we turn off the software cache prefetching by letting the simulator ignore software prefetching instructions, so that we can compare how well AMB prefetching works with and without software cache prefetching.

Figure 12 compares the SMT speedup of three designs, namely AP (AMB prefetching), SP (software prefetching) and AP+SP (the combination) normalized to the SMT speedup of no prefetching at all. For simplicity, we only show the average of all workloads. We have following observations. First, software cache prefetching (SP) does improve the performance significantly. Its speedup is greater than that of AMB prefetching (AP) alone on single, dual and four cores. However, the

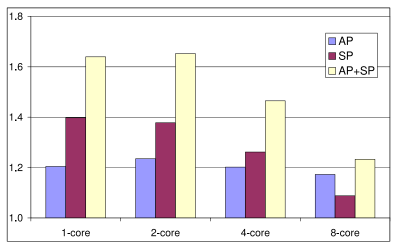

Figure 12: Relative SMT speedup. The complementary between software prefetch and AMB prefetching.

speedup drops with the increase on number of cores. With eight cores, the speedup is less than that of AMB prefetching. Second, the speedup by using both the AMB prefetching and software prefetching (AP+SP) is very close to the sum of SP and AP on single, two, four and eight cores. In other words, the two prefetchings are complementary to each other, although they are not fully orthogonal. We speculate two reasons for this scenario: Bank conflict is only reduced by AMB prefetching, and AMB prefetching may improve the timeliness of software prefetching [20]. We believe that AMB prefetching will improve performance similarly if hardware prefetching is used. We do not include the experiment in this study because it is challenging to fairly evaluate hardware prefetching: There are many design variants [23] and some may incur excessive memory traffic in the worst case.

5.5 Power Saving of AMB Prefetching

The AMB prefetching has a secondary, good effect on power saving: When a memory access hits in the AMB cache, the power for DRAM activation and column access is saved. To estimate the power consumption of memory devices (DRAM chips), we count the number of row and column accesses and use an existing calculator [14] of estimating the power consumption of each pair of activation/precharge and each column access. By feeding the DDR2 parameters to the calculator and assuming 70% bandwidth utilization and 0% row buffer hit rate (using the close page mode), we estimate the ratio of power consumption between the activation/precharge and column access to be roughly 4:1. Activation and precharge are separate operations but their numbers are almost equal under the close page mode with auto precharge. We can use this ratio to calculate the ratio of power consumption with and without AMB prefetching, for which our simulator tells the

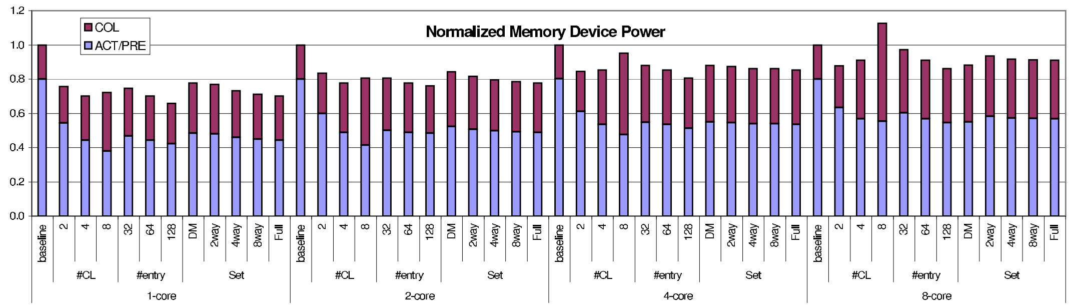

Figure 13: Power saving for FB-DIMM with AMB prefetching.

number of activation/precharge and the number of column access. Note that the tag and data parts of the AMB-cache are accessed in two steps: The tag array in the memory controller is searched first and then the data block in the AMB is accessed. The second step is not associative search and does not incur extra power consumption related to associative search.

Figure 13 shows the power consumption at all memory devices with AMB prefetching variants, all normalized to that of the FB-DIMM without AMB prefetching. The estimated power consumption only includes the dynamic power of the memory devices, not the static and terminal power. According to the calculator [14], the static power is about 17.5% of the total power consumption for the above configuration. The power consumed by the FB-DIMM channels and AMBs is not included because their degree of activities is almost the same with or without the AMB prefetching. The power of the AMB-cache is not estimated either, because it is a very small component when compared with DRAM activation, column access and DDR2 bus transaction. A final note is that the figure does not show the energy saving of the processor: As AMB prefetching improves performance, it also reduces processor execution time and energy consumption.

As the figure shows, in general, the AMB prefetching can significantly reduce the power consumption of memory devices. For instance, it can save 29.9% and 14.7% of power consumption for single-core and four-core workloads using the four-cacheline interleaving, respectively. Because no row and precharge operations are required for prefetch buffer hits and row and precharge operations consume much more power than column accesses, the AMB prefetching can save the total power consumption even if it issues some unnecessary column accesses. Grouping more cache lines together can further reduce the number of row accesses but at the cost of more unnecessary column accesses. For example, for four-core workloads, the number of ACT/PRE accesses decreases by 24.0%, 33.3% and 40.6% for one, two and eight-cacheline interleaving, respectively. At same time, the number of column accesses increases by 19.5%, 41.2% and 61.1%, respectively. In the extreme case, for eight-core workloads, using eight-cacheline interleaving will increase the total memory device power consumption by 12.7%. Thus, the AMB prefetching must make a balance between reducing the number of row accesses and increasing the number of column accesses. In addition, increasing the size or associativity of the prefetch buffer may save more power of memory devices because of higher prefetch buffer hit ratio. However, this will increase the power consumption on the prefetch buffer. We plan to study the trade off in the future. In general, to balance between the power consumption and performance, the memory mapping policy and the prefetch buffer configuration need to be carefully considered. For our experimental setup, the prefetch buffer with four-way associativity, 64 cache lines and using four-cacheline interleaving mode is a good choice.

6 Related Work

The FB-DIMM technology is proposed recently and has been adopted by the Intel Corp [24]. Nasr used synthetic workloads to evaluate FB-DIMM technology [16]. Davis et al. used FB-DIMM in their CMT study [5], though did not evaluate FB-DIMM itself. Parulkar and Cypher found that the error tolerance feature in FB-DIMM makes it more robust than others [17]. Recently Ganesh et al. evaluated FB-DIMM in detail [6]. To our best knowledge, this paper is the first work that study DRAM-level prefetching for FB-DIMM.

This work is closely related to DRAM latency reduction techniques. Burger et al. evaluated the latency-reducing techniques such as non-block cache, out-of-order execution, prefetching, and their demand for high memory bandwidth [2]. They concluded that the processor pin bandwidth would be the bottleneck of future processors. Cuppu et al. compared the performance of several DRAM technologies, including Fast Page Mode, Extended Data Out, Synchronous, Enhanced Synchronous, Synchronous Link, Rambus and Direct Rambus [3]. Zhu and Zhang studied DRAM memory access scheduling of SMT processor [28].

There have been many studies in hardware or software cache prefetching [11, 15, 23]. Because of the volume of those existing studies, we cannot fully cite even the most important ones. Those prefetching schemes can be evaluated by their coverage, accuracy, and timeliness [20]. In general, those conventional prefetching schemes may reduce idle memory latency much better than DRAM-level prefetching. Additionally, their coverage may be better because they may observe full cache miss addresses to predict future misses. However, inaccurate prefetch is unavoidable and will increase memory traffic. The increase of memory traffic is particularly an issue of using those prefetching for multicore processor. A special case of prefetching is to fetch data from DRAM to memory controller [13]. This type is closer to cache prefetching than DRAM-level prefetching because the memory controller and the CPU is on the same side of the memory bus.

DRAM-level prefetching in this paper refers to the prefetching of data at the same side of DRAM memories to the memory bus. In previous studies, the DRAM row buffer is enhanced as a natural buffer, or SRAM cache is added to the DRAM in the case of cached DRAM [10, 7, 12, 27]. The prefetching is actually inside DRAM chips, which enjoys the huge internal bandwidth of DRAM, roughly hundreds of GB/s. AMB prefetching is different in that the prefetching is across the DRAM chip boundary, and thus the available bandwidth is a small multiple of the memory channel bandwidth. The prefetching must be more careful as less redundant bandwidth is available. David [4] has proposed to use a cache module with each memory module for prefetching, which is similar to this work in principle, but no quantitative evaluation was done at that time.

7 Conclusions

In this study, we have studied the DRAM-level prefetching for FB-DIMM that prefetches data from DRAM chips to the AMBs of FB-DIMM. We study the design issues in this class of prefetching and propose a region-based memory prefetching scheme as the implementation. The experiment results show that the proposed AMB prefetching significantly improve the overall performance of single-core and multicore workloads, and the improvement comes from idle memory latency reduction and improved bandwidth utilization. Additionally, the proposed AMB prefetching reduces DRAM power consumption and works well with software cache prefetching.

Acknowledgment

We appreciate the constructive comments from the anonymous reviewers and thank Bruce Christenson at Intel for his critical comments.

References

[1] N. L. Binkert, E. G. Hallnor, and S. K. Reinhardt. Network-oriented full-system simulation using M5. In *Proceedings of the Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads*, 2003.

[2] D. Burger, J. R. Goodman, and A. Kägi. Memory bandwidth limitations of future microprocessors. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 78–89, 1996.

[3] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. A performance comparison of contemporary DRAM architectures. In *Proceedings of the 26th International Symposium on Computer Architecture*, pages 222–233, 1999.

[4] H. David. Distributed memory module cache prefetch. US patent #6,925,534, Intel Corporation, 2001.

[5] J. D. Davis, J. Laudon, and K. Olukotun. Maximizing CMP throughput with mediocre cores. In *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, pages 51–62, 2005.

[6] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob. Fully-buffered DIMM memory architectures: Understanding mechanisms, overheads and scaling. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture*, Phoenix AZ, 2007.

[7] C. A. Hart. CDRAM in a unified memory architecture. In *Proceedings of CompCon'94*, pages 261–266, Los Alamitos, CA, 1994.

[8] J. L. Henning. SPEC CPU2000: measuring CPU performance in the new millennium. *IEEE Computer*, 33(7):28–35, July 2000.

[9] H. Hidaka, Y. Matsuda, M. Asakura, and K. Fujishima. The cache DRAM architecture: A DRAM with an on-chip cache memory. *IEEE Micro*, 10(2):14–25, 1990.

[10] W.-C. Hsu and J. E. Smith. Performance of cached DRAM organizations in vector supercomputers. In *Proceedings of the 20th International Symposium on Computer Architecture*, pages 327–336, 1993.

[11] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th International Symposium on Computer Architecture*, pages 364–373, 1990.

[12] R. P. Koganti and G. Kedem. WCDRAM: a fully associative integrated Cached-DRAM with wide cache lines. In *Proceedings of the 4th IEEE Workshop on the Architecture and Implementation of High Performance Communication Systems*, 1997.

[13] W. Lin, S. Reinhardt, and D. Burger. Reducing DRAM latencies with an integrated memory hierarchy design. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, 2001.

[14] Micron. DDR2 SDRAM system-power calculator. `http://www.micron.com/support/designsupport/tools/powercalc/powercalc`.

[15] T. C. Mowry, M. S. Lam, and A. Gupta. Design and evaluation of a compiler algorithm for prefetching. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 62–73, 1992.

[16] R. Nasr. FBsim and the Fully Buffered DIMM memory system architecture. Master's thesis, University of Maryland, 2005.

[17] I. Parulkar and R. Cypher. Trends and trade-offs in designing highly robust throughput computing oriented chips and systems. In *11th IEEE International On-Line Testing Symposium*, pages 74–77, 2005.

[18] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *Proceedings of the 27th International Symposium on Computer Architecture*, pages 128–138, 2000.

[19] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, 2002.

[20] T. Sherwood, S. Sair, and B. Calder. Predictor-directed stream buffers. In *Proceedings of the 33rd International Symposium on Microarchitecture*, pages 42–53, 2000.

[21] A. Snavely, D. M. Tullsen, and G. Voelker. Symbiotic jobscheduling with priorities for a simultaneous multithreading processor. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 66–76, 2002.

[22] Standard Performance Evaluation Corporation. *SPEC CPU2000*. http://www.spec.org.

[23] S. P. Vanderwiel and D. J. Lilja. Data prefetch mechanisms. *ACM Computing Surveys*, 32(2):174–199, June 2000.

[24] P. Vogt and J. Haas. Fully-Buffered DIMM technology moves enterprise platforms to the next level. `http://www.intel.com/technology/magazine/computing/fully-buffered-dimm-0305.htm`, 2005.

[25] C. Weaver. *http://www.simplescalar.org/spec2000.html*. SPEC2000 binaries.

[26] Z. Zhang, Z. Zhu, and X. Zhang. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *Proceedings of the 33rd International Symposium on Microarchitecture*, pages 32–41, 2000.

[27] Z. Zhang, Z. Zhu, and X. Zhang. Cached DRAM: A simple and effective technique for memory access latency reduction on ILP processors. *IEEE Micro*, 21(4):22–32, July/August 2001.

[28] Z. Zhu and Z. Zhang. A performance comparison of DRAM memory system optimizations for SMT processors. In *Proceedings of the 11th International Conference on High-Performance Computer Architecture*, pages 213–224, 2005.