International Journal of Hybrid Information Technology
Vol.9, No.4 (2016), pp. 319-338
http://dx.doi.org/10.14257/ijhit.2016.9.4.27

A Study of Leveraging Memory Level Parallelism for DRAM System on Multi-Core Architecture

Yuxuan Wang[1], Yingping Zhang[2], Xiaotian Zhang[3], Jian Yin[4] and Licheng Chen[5]

[1,3,4]*Department of Computer, Shandong University, Weihai, China*
[2]*Hunan Electric Power Company, State Grid, China*
[5]*Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*
jianyinsdu@126.com, bryantclc@gmail.com

Abstract

DRAM system has been more and more critical on modern multi-core architecture where the Moore's law has been made effect on increasing the number of cores integrated in a processor chip. The performance of DRAM system is usually measured in term of bandwidth and latency, which are regarded as inherently depending on Row Buffer Hit Rate (RBHR) according to previous studies. In this paper, we find that Memory Level Parallelism (MLP) exhibits a stronger correlation with the performance of DRAM system on multi-core/many-core architecture than RBHR, and promoting MLP significantly improves DRAM system performance. In order to exploit the MLP, we have evaluated various approaches including multi-bank, multi-row-buffers, multi-memory-controllers and the obsolete Virtual Channel Memory (VCM). The experimental results show that VCM is a better alternative to traditional DRAM chip on multicore/many-core architecture than the other three approaches because VCM has almost all the advantages of the others: 1) it can improve homogeneous workloads' IPC by 2.21X on a 16-core system with 32 virtual channels due to leveraging unexploited MLP. 2) It can also promote Quality-of-Service (QoS) of DRAM system by removing unfairness while memory controllers serve memory requests. 3) It can save energy and has low area costs. Unfortunately, VCM, which was proposed in the late 1990s, faded away before multi-core/manycore became dominated. Therefore, we suggest memory chip vendors reconsider the VCM technology for multi-core architecture.

***Keywords**: DRAM; Virtual Channel Memory; Memory Level Parallelism; Qos*

1. Introduction

"MLP yes! ILP no!" Memory Level Parallelism (MLP) was originally proposed in term of the number of outstanding cache misses by Andrew Glew [8], in order to persuade people to do research that helps to exploit MLP. Subsequently, numerous previous studies investigated microarchitectures to enhance MLP from on-chip (processor) side, such as MLPaware cache replacement [24], MLP-aware prefetcher [7, 14] and runahead execution [5]. In these studies, due to limited number of cores and limited parallelism resource (such as Instruction Window, MSHR) on chip, processor was the bottleneck to exploit MLP. But this is not right for multi-core architecture any more, with the rapid increasing number of cores, the shared memory system suffers heavy pressure to service requests form all cores. Thus for multi-core/many-core architecture, the memory system has become the main bottleneck to exploit MLP due to its relative slowly increasing parallelism resource (channel, rank, and bank). The "Memory Wall" problem under multi-core/many-core architecture becomes more and more serious.

To moderate "Memory Wall" problem, contemporary servers would adopt high memory configuration, which could provide high memory bandwidth and MLP. For example, POWER7 processor integrates 8 cores with 4 threads each and two 4-channel DDR3 memory controllers, which could provide as high as 100GBps memory bandwidth if all the channels are fully exploited. However, due to cost and power budget limitation, a large part of servers were not configured with full memory DIMMs/channels exploited. A statistical data from a server vendor company showed that, during 2011, the most popular server sold is configured with 2 sockets, each socket has 6 cores with 6 memory DIMM slots, among these, only 15% of the servers are sold with full DIMMs exploited, 50% with half DIMMs exploited, and 35% with less than half DIMMs exploited. Thus, with the budget limitation, it becomes more important to improve memory bandwidth efficiency and exploit MLP to shorten the memory wall gap.

However, from the DRAM memory system side, because each DRAM bank is integrated with only one 4KB or 8KB row-buffer (or sense-amplifier) which holds data from a 100xMB DRAM array, the Row Buffer Hit Rate (RBHR) is considered as a key factor of the performance of DRAM system. Most researchers have made significant contributions on reducing RBHR by memory access scheduling [27], address mapping and so on. During the past years, the prevalence of multi-core/many-core architecture poses new challenges of performance, power and QoS to DRAM system which is shared by all cores.

Recent studies have proposed a number of solutions to address the challenges caused by multi-core system [6, 18, 20, 21, 23, 25, 26, 37]. Nevertheless, most of the studies were motivated by improving row buffer hit rate (RBHT) of DRAM system, without considering the method of enhancing MLP for DRAM system. Although previous studies show the strong correlation between RBHR and the performance of DRAM system, we find that MLP has even a stronger correlation with the performance of DRAM system than RBHR in multi-core/many-core architecture. Experimental results show that the average IPC of homogeneous multi-program workloads on a 16-core system improves by 1.95X when incrementing the DRAM banks from 8 to 32, whereas the RBHRs are almost the same for the 8-bank and 32-bank DRAM configurations, while the banklevel MLP increases by about 2.00X (please refer to section II for detail).

On multi-core architecture, each core can generate an independent memory request stream. Memory controllers are responsible for scheduling the requests to the available DRAM banks. If there are no available banks, the requests have to queue in the request buffer of memory controller. A recent study has shown that the queuing-delay has become the dominant portion of one memory request’s access latency on multicore system [31]. Given the provision of more available banks, more memory requests would be scheduled, which means there is still a large amount of unexploited MLP due to memory limited parallel resource in multi-core system.

There are several approaches to enhance MLP. As mentioned above, simply incrementing bank count within a DRAM chip is a straightforward method to enhance MLP. There are other methods to enhance MLP, such as using multiple memory controllers, splitting bank into sub-banks and incrementing the number of row buffers. For example, using multiple memory controllers is a widely used method to improve DRAM performance on multicore architecture, but the number of memory controllers is not scalable due to the limited chip pin count. Udipi et al. [31] proposed two new DRAM organizations which contain a large number of subbanks (or sub-arrays) and show performance improvement by 54%. However, these aggressive approaches substantially change the DRAM organization, thereby cause significant redesign cost and high risks.

In this paper, we have evaluated four representative straightforward approaches to leverage MLP for DRAM system on multi-core architecture: 1) multi-bank, 2) multi-rowbuffer1), 3) multi-memory-controller and 4) Virtual Channel Memory (VCM). VCM is selected because it represents a method of providing additional cache on the DRAM chip. Furthermore, alike multi-memory-controller, VCM is a mature technology because it possessed a certain market after it was first introduced by NEC corporation in late 1990s [22], but it faded away later (For more details, please refer to section III).

In order to exploit MLP, VCM might be an ideal alternative to traditional DRAM chip on multi-core/many-core architecture with regard to performance, QoS, power and area overheads and even design cost and risky. Unfortunately, VCM had been obsolete before we entered the multicore/many-core era. Therefore, we suggest memory chip vendors reconsider the VCM technology for multi-core architecture.

Overall, we have made the following contributions:

• We find that MLP has a stronger correlation with the performance of DRAM system on multi-core/many-core architecture than RBHR which is considered as the mainly inherent metric to measure the performance of traditional DRAM system.

• In order to leverage the unexploited MLP existing in multi-core system, we have selected and evaluated four representative approaches' characteristics in term of performance, QoS, power overhead and area cost.

• According to the experimental results, we find that the obsolete VCM exhibits better than the other three approaches. We argue that memory chip vendors could reconsider the VCM technology for multi-core/many-core architecture.

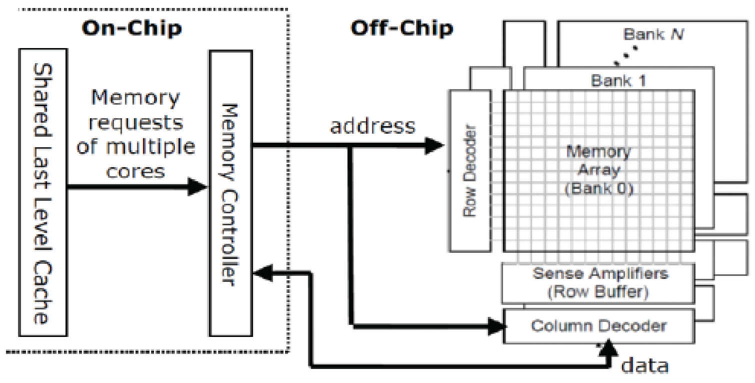


Figure 1. DRAM System Organization

The rest of the paper is organized as follows. In Section II, we introduce the observations of MLP and RBHR on multi- core/many-core architecture. In Section III, we present our evaluation scheme. We describe the experimental setup in Section IV and demonstrate experimental results and discussion in Section V. Related work and conclusion are in Section VI and Section VII respectively.

2. Background and Motivation

2.1. DRAM Memory System

Figure 1 illustrates the organization of DRAM system. Contemporary multi-core processors often integrate one or more memory controllers on the chip. Each memory controller consists of 1-3 memory channels. Since adding memory channels essentially has the same effect of adding memory controllers, we use one-channel per memory controller and one-rank per channel in this paper for simplicity. So each memory controller manages multiple (usually eight) DRAM banks which can independently process multiple outstanding memory requests in parallel. Each bank is organized as a two-dimensional array of DRAM cells, consisting of multiple rows and columns. These cells are thus accessed using a DRAM address of <bank, row, column> fields, but only one row in a bank can be accessed at any given time. This row requires being stored in the row-buffer (or sense amplifier) before it could be read or written. Each bank of modern DRAM chip has only one row buffer whose size is typically 4-16KB.

If one memory request misses in the row buffer, a row buffer conflict occurs. Then the memory controller issues a PRECHARGE command to update the row in the row buffer back into the memory array, and then issues an ACTIVE command to fetch a new row into the row buffer. Therefore, the row buffer conflict causes significant memory access delay, and degrades system performance. During the past decade, numerous studies have investigated on how to improve Row Buffer Hit Rate (RBHR).

On the other hand, it is vital to keep as many banks busy as possible to improve the performance of DRAM system. This is an intuitive method for exploiting MLP. For traditional DRAM system, the maximum MLP is limited to the bank count. There is a notion called DRAM Bank-Level Parallelism (BLP) which indicates the number of multiple requests being served in parallel in different DRAM banks.

2.2. MLP on Multi-Core Architecture

Previous studies have shown the strong correlation between RBHR and DRAM system performance and have proposed a number of approaches to improve RBHR during the past decade. For example, Rixner et al. [27] proposed FR-FCFS scheduling scheme which prioritized those memory requests hitting in row buffer. Recent studies investigated the RBHR on multi-core architecture. Udipi et al. [31] illustrated that

RBHR decreases significantly from 1 core (over 60%) to 16 cores (35%) mainly due to the row buffer conflicts caused by memory requests from different cores interfering with each other. Sudan et al. also observed the same RBHR trend in their work.

In this paper, we have investigated both MLP and RBHR characterization for multi-core/many-core architecture on our simulation platform. Figure 2 shows the MLP trend in term of Normalized BLP on a 32-bank2) memory system as the number of cores is varied from 1 to 16, where the baseline is 1-core system. Here we ran homogeneous multi-program workloads with each core ran the same program from SPEC-CPU2006 benchmark.

We can see that except for 400.perlbench, 403.gcc, 429.mcf, 444.namd, 447.dealII, 458.sjeng, and 471.omnetpp formed workloads, all the other workloads has the normalized BLP larger than 5 with 16-core, and the geometric mean of all the workloads is 6.16 with 16-core. We can also see that the geometric mean of BLP is increasing proportionally to the number of cores (or threads) increasing. The result shows that with the number of cores increasing, the demand of MLP increasing proportionally, which would put a heavy pressure on traditional DRAM system. The least increasing of normalized BLP is 400.perlbench, it is only 1.54 with 16-core. That is because it is a memory non-intensive (the Last Level Cache MPKI is only 0.04 on 1-core) program, even with 16-core running in parallel, it still fails to exploit MLP due to its rare memory requests. But for 462.libquantum, which has the most increasing rate of normalized BLP, achieves 16.07 in 16-core, that is because it is memory-intensive and having quite good memory locality (the RBHR of it is 91.50% in 1-core). In our simulation, we adopt the bank-interleave address mapping scheme for exploiting BLP, which means we map the least bits of cache block address for bank identity. The contiguous memory accesses are mapped interleaved into multiple banks (thus exploit BLP).

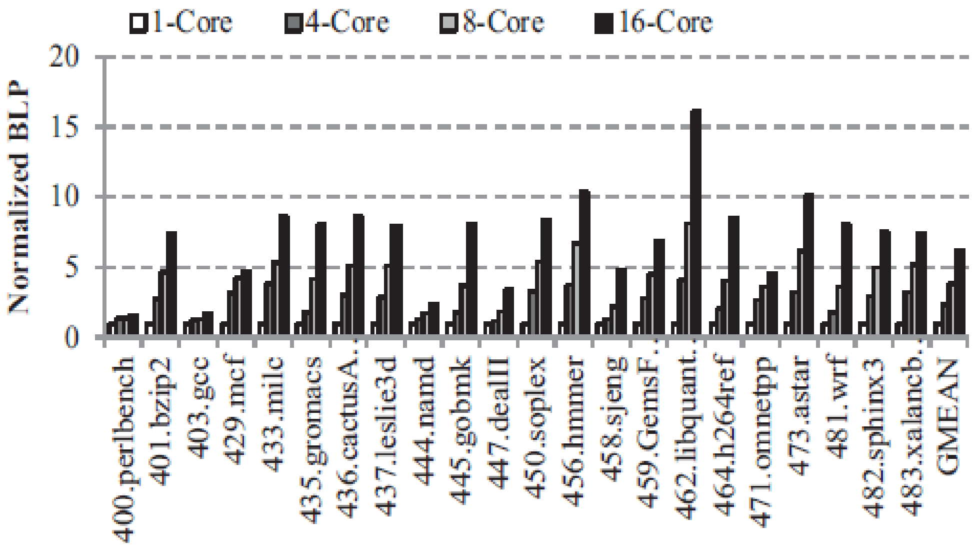


Figure 2. The Normalized BLP (Bank Level Parallelism) Trend on a 32-Bank Memory System as the Number of Cores is varied from 1 to 16, where the Baseline is 1-core

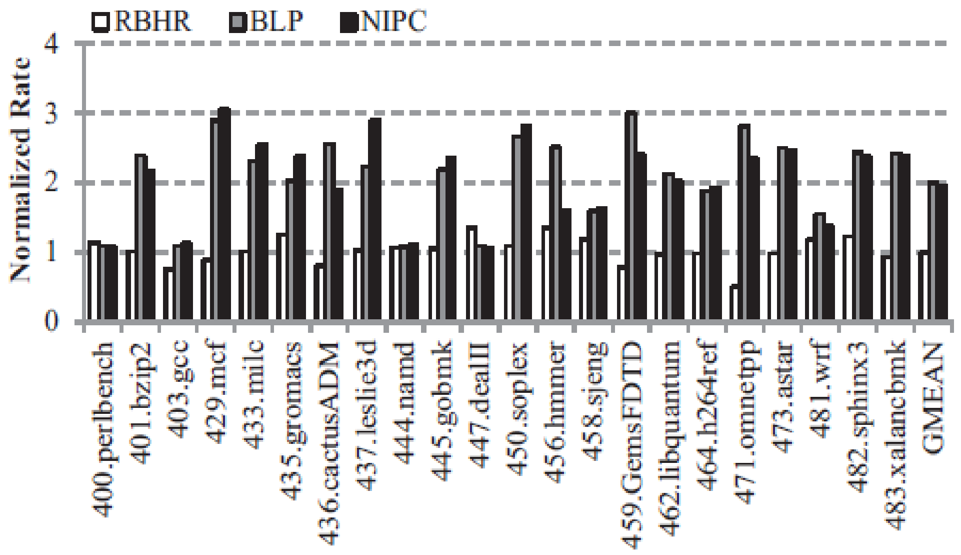


Figure 3. The Normalized Rate of RBHR (Row Buffer Hit Rate), BLP and NIPC (Normalized Instructions Per Cycle) with 32-bank Memory on a 16-core System, where the Baseline is 8-bank Memory on a 16-core system

Multi-core architecture poses not only the negative problems (e.g., the memory contention and unfairness problem) but also exposes large amount of MLP which is the aggregation of multiple independent memory request streams generated by multiple cores. Incrementing the bank count is a straightforward approach to exploit MLP. Figure 3 illustrates that on a 16-core system, the Normalized Rate of RBHR (Row Buffer Hit Rate), BLP (Bank Level Parallelism) and NIPC (Normalized Instructions Per Cycle) with 32-bank memory system, where the baseline is 8-bank setup. The 32-bank setup can exploit BLP nearly 2 times more than the 8-bank setup, thereby improve overall system performance by nearly 1.94 in term of normalized IPC. We can also see that the more BLP exploited the more IPC speedup achieved. On the other side, the Normalized RBHR of 32-bank setup is almost equal with 8-

bank setup. For some workloads (such as 403.gcc, 459.GemsFDTD, 471.omnetpp), the normalized RBHR even decreased for 32-bank compared with 8-bank. The most amount of decreased workload is 471.omnetpp, the normalized RBHR is only 0.50X of the 8-bank setup. The probable reason is that with bank-interleave address mapping, the memory requests from 16-core mixed and interfered with each other, thus further decreased row buffer hits. But for 471.omnetpp, the normalized BLP speedup achieved at 2.80, which could brought the improvement of normalized IPC by 2.35 even with worse RBHR. Based on these observations, we can conclude that MLP has a stronger correlation with the performance of DRAM system than RBHR on future multicore/many-core architecture. Leveraging MLP could effectively improve system performance.

However, since the memory chip vendors focus on costper-bit and device density, they would not like to increase the number of bank because adding more banks means requiring more resources for additional sets of row decoders, sense amplifiers and column muxes etc. Actually, the number of banks integrated in a DRAM chip did not change too much in the past decade, from 4 banks in DDR SDRAM to 8 banks in DDR3 SDRAM. As the number of cores increases more and more, the limited bank count leads to a large amount of unexploited MLP.

3. Leveraging MLP

There are two design philosophies for leveraging MLP. One is "design from scratch", which means substantially changing DRAM organization. Several recent studies, e.g., Selective Bitline Activation (SBA) and Single Subarray Access (SSA) [31], have proposed new DRAM organizations to improve DRAM system performance by enhancing MLP. However, although this design philosophy might bring significant changes, it also might lead to unpredictable outcome and high risks. Another design philosophy is "keep it simple and stupid (KISS)", which means looking for approaches that are either already existing or combinations of existing technologies.

In this paper, we adopt the KISS design philosophy to investigate how to leverage MLP. We select four straightforward approaches:

Multi-Row-Buffer: To keep the design simple, we only increment the row buffer count but do not change the DRAM state transition diagram, parameter and specifications. Therefore, although those row buffers hold multiple opened rows, only one row is accessible at any time according to the DRAM specification. Actually, this approach can improve RBHR other than enhance MLP, so it can be used to compare the effectiveness of improving RBHR and enhancing MLP.

Multi-Bank: Given a capacity-fixed DRAM chip, we split it into different number of banks. This approach requires additional resources for memory controller (multiple control logic modules) and DRAM chip (address decoders and row buffers etc.), but it does not need to change the DRAM specification.

Multi-Memory-Controller: We increase the number of on-chip memory controllers. This approach inherently increases the number of banks and should have the same effect as multi-bank. However, it consumes on-chip resources, especially the pin count.

Virtual-Channel-Memory (VCM): VCM represents a method of providing additional cache on the DRAM chip. In each rank, there are 16 to 32 channel buffers, each holding one segment of row buffer. The DRAM specification is slightly changed to support operating channel buffers, but VCM memory controller is compatible to traditional DRAM. Furthermore, VCM is a mature technology and ever possessed a certain market around 2000.

Since VCM requires changes to DRAM specification, we would like to describe it in details. VCM was first introduced by NEC corporation in late 1990s [22]. It was intended for a wide range of applications such as multimedia and web servers. VCM puts a set of fast channel buffers within memory chips and the number of channel buffers is usually 4 or 8 times more than that of banks. Hence, VCM is expected to provide faster access as well as more concurrency.

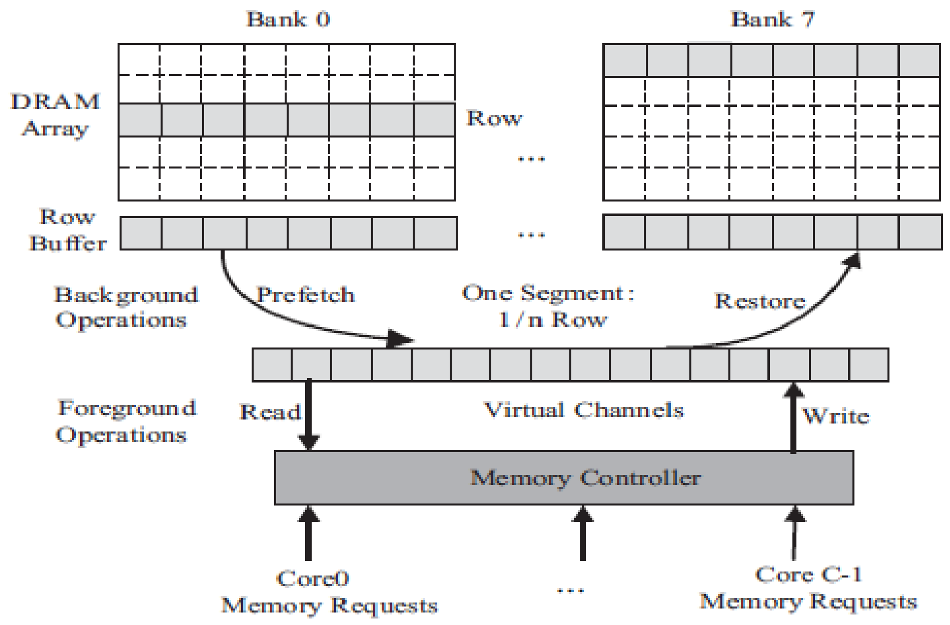


Figure 4. Conceptual Organization of VCM

3.1. VCM Organization

Figure 4 illustrates VCM's conceptual organization. Channel buffers are introduced as an extra storage layer between memory controller and DRAM banks. Two commands, i.e., PREFETCH (reading segment data from row buffer to channel) and RESTORE (writing segment data from channel to row buffer), are also introduced in order to operate channels. As shown in the figure, each row buffer is divided into 4-16 segments which are transfer units between banks and channels. Memory operations (commands) are divided into foreground operations for channels (READ and WRITE commands) and background operations for DRAM banks (ACTIVE, PRECHARGE, PREFETCH and RESTORE commands). NEC's VCM is implemented to be compatible to the industry standard SDRAM and uses the same command protocol and interface as SDRAM/DRAM. Because channels and DRAM banks are independent, foreground operations and background operations can also be executed independently.

To further enhance VCM's performance, channels and the row buffers (banks) of the original VCM are fully associative, which means that any channels can store segment data from any row buffers (banks) and can be written back to any row buffers (banks). Because channels and DRAM banks are independent, memory controller can schedule the foreground and background operations concurrently, which allows the memory controller to exploit as more MLP as possible.

3.2. VCM for Multicore Architecture

We have implemented VCM on our simulation platform. Figure 5 illustrates the design of VCM memory controller which is responsible for scheduling memory requests from different cores. There are three main distinctions between a VCM memory controller and a traditional DRAM controller:

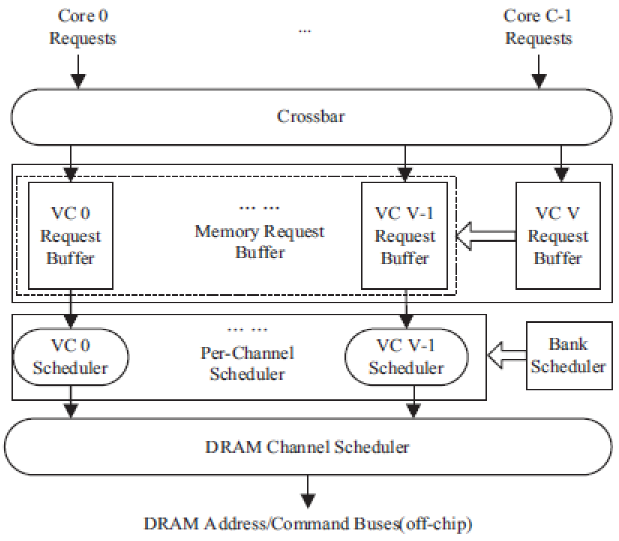


Figure 5. VCM Memory Controller

1. The channels are decoupled with bank, which means that the channels can hold any segment data from any bank/row. Thus each channel needs a tag register to record memory address of the segment it holds (i.e., <bank, row, segment>).

2. There are channel-level schedulers to schedule VCM commands to channels. Those schedulers need to keep track of their corresponding channels' state to determine which command should be issued.

3. There are bank-level schedulers for keeping track of the state of the banks. The bank state information and the channel state information are used to select requests in request buffer.

NEC's VCM usually has 8 to 32 channels. In fact, channel buffers operate like a fully associative cache with write through policy. Take reading data from a memory bank as an example. Memory controller first looks up registers to determine whether the required data is already in a channel; if the read request misses in all channels, the bank-level controller will issue an Active command to the corresponding bank to fetch a row into the row buffer; after 2 (tPAD) cycles, a Prefetch command issued by a channel-level scheduler follows to transfer one segment of the row buffer to a channel which memory controller selects for replacing; then 2 (tPCD) cycles later, the Read command can be issued to the new refilled channel and after another 2 cycles (read latency) the data can be present in the data bus; meanwhile, if using a close-page policy, a Precharge command is issued to the bank to close the row; for an 8 burst length, the data transfer requires 4 cycles with double data rate (DDR). As for writing data into a channel buffer, Write, Restore and Active commands are issued consecutively , and finally a Precharge command is used to close the corresponding row. According to the datasheet [22], we conclude that the original NEC's VCMs adopt write-through policy for channel buffers and close-page policy for row buffer. For more details of VCM, please refer to [26]3).

3.3. Optimization for Contention and QoS

Like the conventional DRAM system, we think that the memory scheduling algorithm is critical for VCM to optimization for contention and QoS in multicore architecture. Here we implement three different scheduling algorithms for DRAM and VCM:

FRFCFS: the FRFCFS [27] scheduling algorithm is probably the most popular and effective one in uniprocessor systems and it is used as a baseline in this paper. In DRAM system, it prioritizes the requests in two levels: (1) Row-hit first; (2) Oldest first.

In the VCM, the FRFCFS algorithm works like:

1. Channel hit first: the controller first searches the idle (the channel is ready for Read/Write) and non-empty channel buffers to see if requests hit. And those hit requests are scheduled with high priority. Note that, for requests hit in channels, there is no need to consider their corresponding bank state, since they only require foreground Read/Write operations and have the least latency.
2. Bank ready first: If no requests hit in channels, the controller chooses a request from the Recycle Request Buffer with ready (or idle) corresponding bank, this procedure needs to cooperate with the bank scheduler. If all the banks in the VCM are busy in background operations, then no memory requests can be issued this time.
3. Oldest first: If there has multiple hit requests or bankready requests at the scheduling time, the oldest request (i.e. requests that arrived earlier in the VCM controller) is prioritized over those younger requests.

PARBS: grouping memory requests into batches and prioritizing older batches over younger ones are adopted to achieve a balance between fairness and throughput. It maintains the maximum number of requests in each bank for each thread to prioritize threads and schedules requests as batches to avoid unfairness. It further serves memory requests in bank-level parallelism.

In the VCM, the VC-PARBS algorithm is almost the same. The VCM controller keeps track of the number of requests in each channel buffer and the Recycle Request Buffer, it makes batch in the channel buffer for each thread, and servers the thread requests in channel-level parallelism.

ATLAS: prioritizing threads that have attained the least service from the memory controllers is adopted [13]. The controller keeps track of the history servicing time of each thread to make scheduling decision.

VC-ATLAS is just the same with ATLAS algorithm, the VCM need additionally keep track of the history servicing time of each thread in all VC buffers with the state of VCMs and banks to make scheduling decision.

4. Experimental Setup

4.1. Evaluation Tools

We implement and evaluate the four approaches using an in house cycle-accurate x86 CMP simulator. The functional front-end is based on Pin [15] and iDNA [4]. We model the memory system in detail, faithfully capturing bandwidth limitations, contention, and enforcing bank/channel/bus conflicts. Table 1 shows the major DRAM and processor parameters. We model a modest multi-core (16 core) system with one channel as baseline to produce heavy access pressure on memory system to simulate the environment in future multi-core/many-core system, meanwhile limiting simulation time. The VCM was implemented in the memory system, the baseline configuration is 32 Virtual Channels within each DRAM chip. We use CACTI 6.5 [1] to evaluate area and power parameters.

Table 1. Simulated System Configuration

Processor Pipeline	3 GHz processor, 128-entry instruction window (64-entry issue/store queue), 12-stage pipeline
Fetch/Exec/Commit width	3 instructions per cycle in each core; only 1 can be a memory operation
L1 Caches	32 K-byte per-core, 4-way set associative, 64-byte block size, 2-cycle latency
L2 Caches	512 K-byte per core, 8-way set associative, 64-byte block size, 12-cycle latency, 32 MSHRs
Baseline DRAM controller	FR-FCFS, close-page row buffer policy for VCM, open-page for other approaches; 128-entry request buffer, 64-entry write data buffer, reads prioritized over writes, XOR-based address-to-bank mapping [33].
DRAM chip parameters	Micron DDR3-1600 [17], tCL=15ns, tRCD=15ns, tRP =15ns; 8 banks, 8K-byte row-buffer per bank
DIMM configuration	Single-rank, 8 DRAM x8 chips to provide a 64-bit wide channel to DRAM
L2 miss latency with VCM	row-buffer hit: 200 cycles, closed: 300 cycles, conflict: 400 cycles. VCM Latency: Active-to-Prefetch (100 cycles), Prefetch-to-Read/Write (80 cycles), Read (200 cycles)
VCM parameters	1K-byte segment, 8 segments per row-buffer, 8 requests VC request buffer, 128 requests Recycle Request Buffer

Table 2. Memory Intensive Benchmark Characteristics

Benchmark	Type	IPC	MPKI	RBHR	BLP
401.bzip2	INT	1.07	3.42	80.93	1.58
429.mcf	INT	0.18	71.86	10.82	5.05
433.milc	FP	0.27	20.59	86.54	1.29
436.cactusADM	FP	0.45	6.31	19.70	1.32
437.leslie3d	FP	0.35	17.10	71.04	1.71
450.soplex	FP	0.19	38.99	88.27	1.71
456.hmmer	INT	0.77	5.19	49.76	1.29
459.GemsFDTD	FP	2.41	3.78	52.70	2.87
462.libquantum	INT	1.25	6.90	91.50	1.03
464.h264ref	INT	3.28	2.39	85.20	1.12
470.lbm	FP	0.56	35.26	85.85	3.31
471.omnetpp	INT	2.10	5.85	62.70	3.56
473.astar	INT	1.18	6.39	51.80	1.47
481.wrf	FP	3.07	1.98	71.39	1.03
482.sphinx3	FP	2.98	2.47	81.97	1.86
483.xalancbmk	INT	2.22	3.70	68.68	2.05

Table 3. Memory Non-Intensive Benchmark Characteristics

Benchmark	Type	IPC	MPKI	RBHR	BLP
400.perlbench	INT	2.05	0.04	68.54	1.33
403.gcc	INT	1.73	0.22	62.12	1.65
435.gromacs	FP	1.55	0.95	80.85	1.43
444.namd	FP	2.44	0.05	91.51	1.05
445.gobmk	INT	1.82	0.60	58.92	1.35
447.dealII	FP	1.85	0.08	83.85	1.17
453.povray	FP	1.88	0.00	88.54	1.58
454.calculix	FP	1.73	0.01	84.70	1.17
458.sjeng	INT	1.94	0.43	24.20	1.54
465.tonto	FP	2.10	0.16	10.85	1.73

4.2. Workloads

We use the SPEC CPU2006 benchmarks for evaluation. We compile each benchmark using gcc 4.1.2 with -O3 optimizations and choose a representative simulation phase using PinPoints [15]. We select memory intensive benchmarks and memory non-intensive benchmarks from SPEC CPU2006. Table 2 and Table 3 list their characteristics (including IPC, MPKI, RBHR and BLP). We run multiple programs on multicore system where each core is dedicated to one program. We use homogeneous workloads (multiple instances of the same program) to evaluate

Copyright ⓒ 2016 SERSC

performance and heterogeneous workloads (the combinations of different programs) to evaluate QoS. All programs are run with their reference (maximum size) input sets. For multi-program workloads: we fast forward 20 million instructions for each process to warm up the simulator, and then execute another 100 million instructions for each core, and then collect simulation data such as IPC, memory access latency and power consumption.

4.3. Metrics

We evaluate the benchmark by four main metrics, i.e., performance, QoS, power consumption and area cost. For individual benchmark, we evaluate performance in term of IPC. For whole system, we use the Unfairness metric to estimate QoS and evaluate performance in term of System_Throughput [12, 29]:

$$Slowdown_i = \frac{IPC_i^{shared}}{IPC_i^{alone}} \tag{1}$$

$$Unfairness = \frac{maximum\{Slowdown_i\}}{minimum\{Slowdown_i\}} \tag{2}$$

$$System\_Throughput = \sum_i Slowdown_i \tag{3}$$

4.4. Experimental Schemes

We adopt three experimental schemes: 1) we use homogeneous workloads to evaluate system performance in term of average IPC. 2) We use heterogeneous workloads to evaluate QoS effect of the approaches in term of Unfairness and System_Throughput. 3)We demonstrate the area cost and power consumption by the CACTI tool.

5. Experimental Results

5.1. Performance

Figure 6 shows the normalized IPC speedup of memoryintensive homogenous workloads running on a baseline multicore system which has 16-core and 8 memory banks. We apply the four approaches to the baseline system and measure their IPC speedups.

For the multi-row-buffer approach, even using eight 8-KB row buffers in one bank, the GMEAN performance improvement is only 1.22X. As mentioned above, this approach can improve RBHR but is still limited to bank-level parallelism. Figure 7 further illustrates the speedups on systems with different number of cores. For the 1-core system, incrementing row-buffer can achieve almost the same improvement as the other approaches. However, as the number of cores increases, its performance scalability is very poor, compared with other approaches. The multi-bank and the multimemory-controller approaches exhibit good IPC speedups. According to Figure 6, 4 memory controllers exhibit the best improvement, by 2.44X while 2 memory controllers setup improves performance by 1.63X. The 32-bank scheme and the 16-bank scheme exhibit performance improvement by 2.44X and 1.63X respectively, and 32-bank can achieve nearly the same improvement as four memory controllers.

On average, VCM with 32 1KB-channel buffers achieves 2.21X performance speedup and also exhibits good scalability, from 1.24X for single-core to 2.21X for 16-core. In fact, an interesting phenomenon is that its RBHR4) is much less than the multi-row-buffer approach. However, according to Figure 8, the MLP (in term of BLP here) improves significantly. Even on a 16-core system with only 8 banks, the VCM with 32-channel can exploit normalized MLP nearly 2.09x compared with 8-Bank. The VCM's capability of exploiting MLP benefits from its organization that channels and DRAM banks are independent so that memory controller can schedule the bank operations and the channel operations concurrently. This phenomenon is strong evidence that the performance on multi-core architecture is more dependent of MLP than RBHR. We further investigate how the latter three approaches exploit more MLP

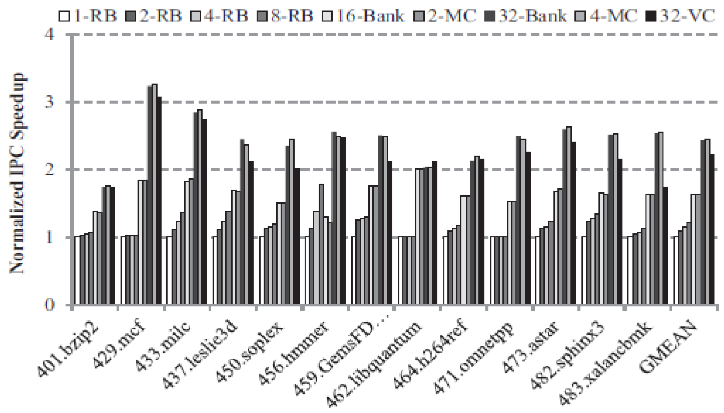


Figure 6. The Normalized IPC Speedup on 16-core System, the Baseline is 8-bank with 1-row Buffer. Here, RB: Row Buffer, MC: Memory Controller

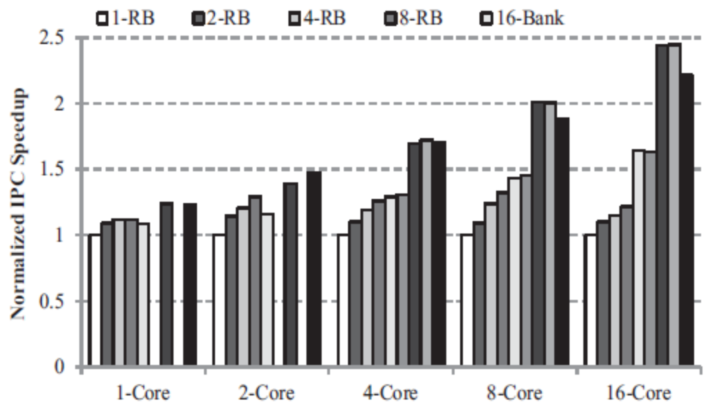
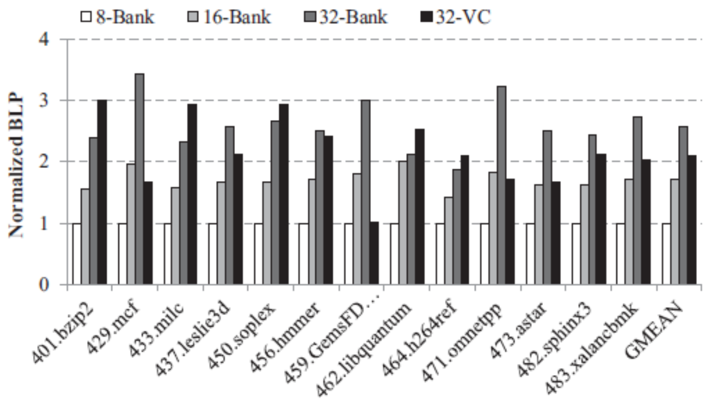


Figure 7. Performance Scalability against the Increasing Core Numbers, in Terms of Normalized IPC



Copyright ⓒ 2016 SERSC

Figure 8. The Normalized BLP (Bank Level Parallelism) on a 16-core System, the Baseline is 8-Bank

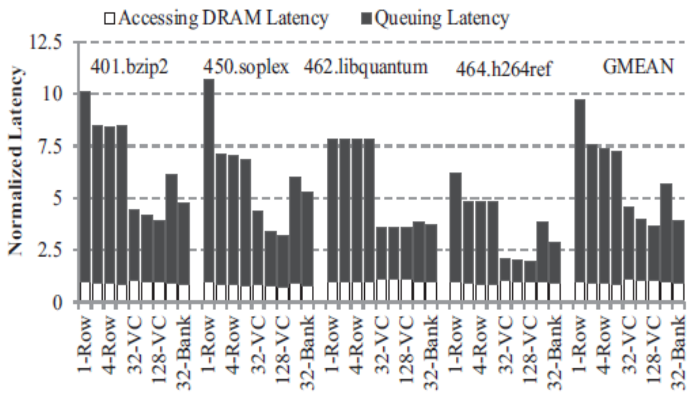


Figure 9. Breakdown of Memory Access Latency on a 16-core System, where theBaseline is Accessing DRAM Latency of 1-Ro

Figure 9 illustrates the average access latency of memory requests. The latency is divided into two parts: DRAM accessing latency and queuing latency. In fact, the DRAM accessing latencies are almost the same for all approaches, however, the queuing delay at memory controller side reduces significantly for the VCM approach and the multi-bank (the same as multi-MC) approach. Take VCM as an example, for each individual request, although the latency of background operations (i.e., accessing DRAM latency) increases slightly by 7.3%, the queuing latency substantially decreases, from 8.75x (of baseline) to 3.50x for 32-channel VCM.

5.2. Impact of VCM Parameters

Figure 10 shows the impact of the number of channel buffers. For most memory intensive benchmarks, 32 channels already exhibit good improvements by about 2.08X. Some benchmarks with poor locality such as 456.hmmer can benefit from more channels. Because their memory access addresses are distributed discretely, row buffer thrilling within a bank is much more likely to occur. While channel number increases, these memory requests with discrete addresses but within one channel can be hit and return immediately without any background operations. As shown in Figure 9, for these benchmarks, the more is channel number, the better are performance improvements.

Figure 11 illustrates the impact of different channel sizes. According to the figure, 32-channel VCM with 2KB channel size has the best performance improvements (2.30X). We investigate the reason and find that large size channel has higher channel hit rate. Take 450.soplex as an example, the channel hit rate for 32 0.5KB channel is 60.0%. While increasing channel size to 1KB and 2KB, the hit rate increases to 68.1% and 74.3% respectively, thereby effectively improving 450.soplex’s performance. For 429.mcf, the hit rates are 22.0% for 0.5KB-channel, 23.8% for 1KB-channel and 24.4% for 2KB-channel. Therefore, there are almost no improvements for 429.mcf while increasing channel size.

To show the VCM could scale well on more aggressive memory system, we configure the 16-core system with at most 4 memory controllers. For each memory controller, there are 4 independent channels, 4 ranks each channel (2 DIMMs per channel, and 2 ranks per DIMM in our simulation), and 8 banks each rank. We implemented VCM in each rank with 32 virtual channels (VC32), the other parameters are just the same as shown in Table 1. We ran 24 randomly selected memory-intensive workloads on a 16-core system.

Figure 12 shows the Normalized System Throughput without and with VCM as the number of memory controller varied from 1 to 4, the baseline is 4 Memory Controllers without VCM (noVC_mc4). We can see that, without VCM, the average performance is only 35.4% for 1 memory controller (noVC_mc1), and 60.0% for 2 memory controllers (noVC_mc2). And with VCM, it could achieve 56.27% for 1 memory controller (VC32_mc1), which improves by 58.95% compared with noVC_mc1; and achieve 81.66% for 2 memory controllers (VC32_mc2), which improves 36.10% compared with noVC_mc2.

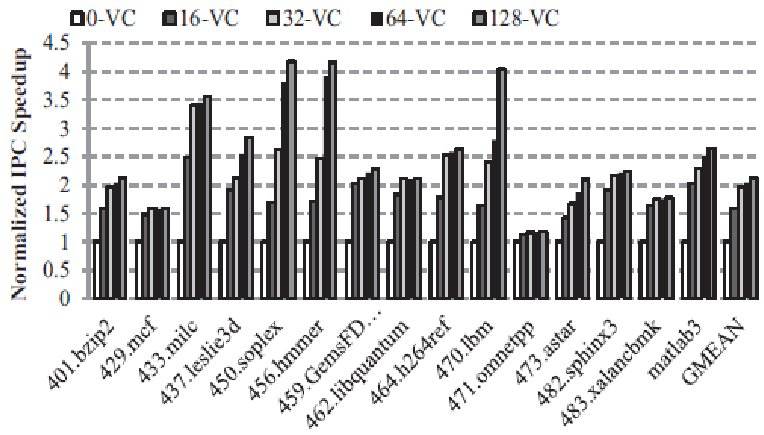


Figure 10. Normalized Performance (IPC) Improvements for Different Number of Virtual Channels on a 16-core System, the Baseline is 0-VC

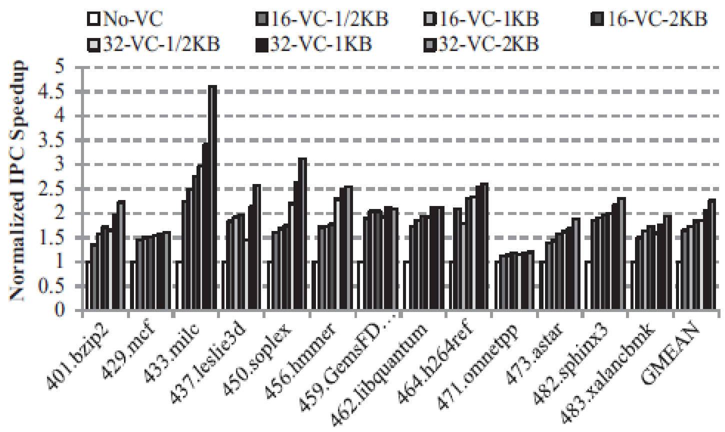


Figure 11. The Normalized Performance Improvements for Different Channel Size of Virtual Channels on a 16-core System, the Baseline is NO-VC

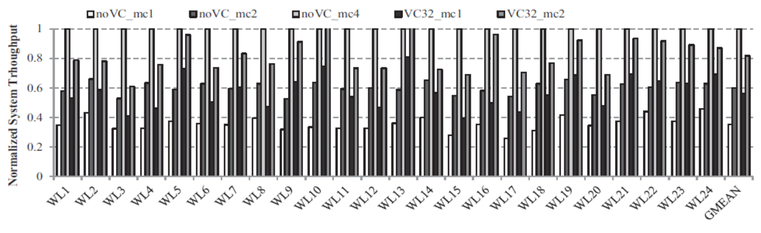


Figure 12. The Normalized System Throughput without and with VCM as the Number of Memory Controllers Increasing from 1 to 4, the Baseline is 1-MC without VC

5.3. QoS

The memory QoS problem means that unfair servicing of different cores' requests by the memory controller can lead to application/core starvation [21, 23] and even denial of memory service for some cores [19]. For example, Mutlu et al. [21] showed that the slowdowns for some memory nonintensive applications can increase from 7.74X for 4-core system to 11.35X for 8-core system whereas the memoryintensive application experienced the slowdowns of only 1.04X and 1.09X respectively. Some other works [18–20,25] also demonstrated the similar unfair phenomenon. Recent studies on memory controller optimization, such as PARBS and ATLAS [13], have shown their effectiveness in improving QoS.

According to our above analysis, VCM is better than the other three approaches for performance. Thus in this section, we mainly evaluate QoS for VCM. Moreover, we have integrated the PAR-BS and ATLAS scheme into VCM. We use the following metrics to evaluate QoS: Unfairness and System_Throughput.

Figure 13(a) shows the average Unfairness and System_Throughput of 16 memory-intensive workloads on a 16-core 8-bank system with and without VCM. Each workload includes 16 memory-intensive programs which are generated by randomly selected from Table 2. We can see that without VCM, the two state of the art scheduling algorithm could efficiently improve system Qos by reducing unfairness from 16.74 to 7.53 (reduce 55.02%) for PARBS and 7.36 (reduce 56.03%) for ATLAS. Meanwhile they can improve System_Throughput by 10.13% for PARBS (4.13 vs. 3.75) and 13.33% for ATLAS (4.25 vs. 3.75). On the other hand, VCM exhibits good performance both in QoS and System_Throughput even with simple memory schedule algorithm (FR-FCFS). It can reduce unfairness by 64.16% (6.00 vs. 16.74) and significantly improve system throughput by 90.67% (7.15 vs. 3.75), both of these are even better than the two state of the art memory scheduling algorithm without VCM. There are two reasons: 1) the fewer memory requests stay in queue, the less is the probability of unfairness. As shown in Figure 9, VCM can significantly reduce queuing latency, so it can also substantially eliminates unfairness; 2) Channels are independent of banks and they can operate simultaneously, hence more memory requests can be issued without the limitation of bank count. Therefore, those requests with low priority have more chances to be scheduled. Although PAR-BS and ATLAS algorithms can largely reduce unfairness by ranking threads' requests, they still subject to the bank-level parallelism. When applying PAR-BS and ATLAS to VCM, the Unfairness metrics change slightly to 6.49 (+8.17%) and 5.99 (-0.17%) respectively, and that is nearly the same for System_Throughput by 6.01% and 0.14%. Thus we can conclude that VCM could perform well with simple FRFCFS memory scheduling algorithm for memory-intensive workloads, further, Figure 10(b) could also prove this view for mixed memory-intensive and memory-non-intensive workloads.

Figure 13(b) shows that for mixed memory-intensive and memory-non-intensive workloads, without VCM the PARBS and ATLAS reduced unfairness quite slightly (3.24% and 3.13%), but the VCM with FRFCFS could also effectively reduce unfairness by 47.18% (29.49 vs. 55.83) and improve system throughput by 66.19% (6.27 vs. 10.42). These results show that VCM could be suitable for various workloads in multi-core/many-core architecture.

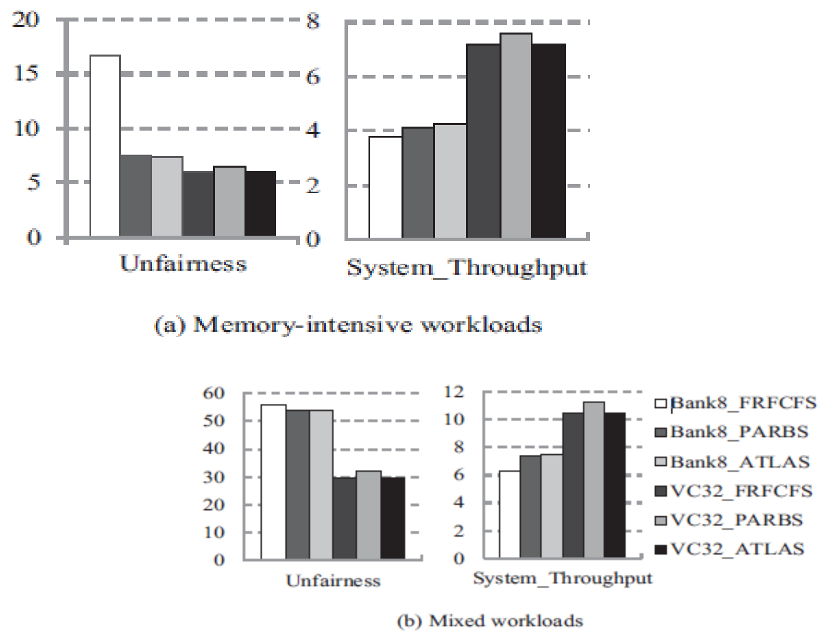
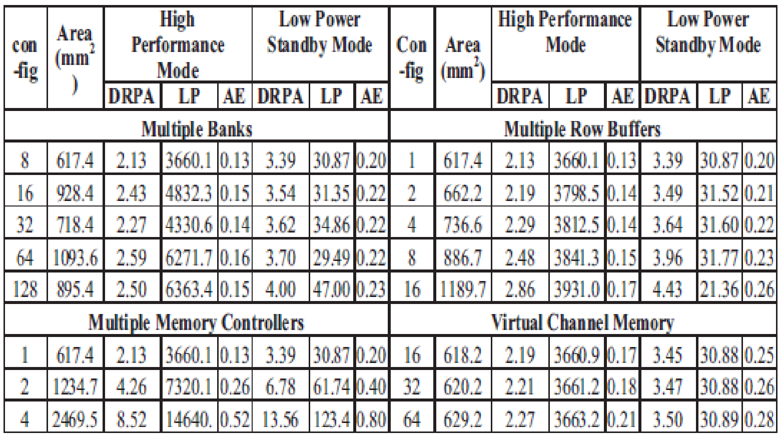


Figure 13. The Unfairness and System Throughput on 16-core System with and without VCM: (a) Memory-intensive Workloads; (b) Mixed Memory-intensive and Memory-non-intensive Workloads



con -fig	Area (mm^2)	High Performance Mode			Low Power Standby Mode			Con -fig	Area (mm^2)	High Performance Mode			Low Power Standby Mode		
		DRPA	LP	AE	DRPA	LP	AE			DRPA	LP	AE	DRPA	LP	AE
Multiple Banks								Multiple Row Buffers							
8	617.4	2.13	3660.1	0.13	3.39	30.87	0.20	1	617.4	2.13	3660.1	0.13	3.39	30.87	0.20
16	928.4	2.43	4832.3	0.15	3.54	31.35	0.22	2	662.2	2.19	3798.5	0.14	3.49	31.52	0.21
32	718.4	2.27	4330.6	0.14	3.62	34.86	0.22	4	736.6	2.29	3812.5	0.14	3.64	31.60	0.22
64	1093.6	2.59	6271.7	0.16	3.70	29.49	0.22	8	886.7	2.48	3841.3	0.15	3.96	31.77	0.23
128	895.4	2.50	6363.4	0.15	4.00	47.00	0.23	16	1189.7	2.86	3931.0	0.17	4.43	21.36	0.26
Multiple Memory Controllers								Virtual Channel Memory							
1	617.4	2.13	3660.1	0.13	3.39	30.87	0.20	16	618.2	2.19	3660.9	0.17	3.45	30.88	0.25
2	1234.7	4.26	7320.1	0.26	6.78	61.74	0.40	32	620.2	2.21	3661.2	0.18	3.47	30.88	0.26
4	2469.5	8.52	14640.	0.52	13.56	123.4	0.80	64	629.2	2.27	3663.2	0.21	3.50	30.89	0.28

Figure 14. Area and Energy Parameters

5.4. Area and Power Cost

We use CACTI to estimate area and power cost for the four approaches. Figure 14 illustrates the area and energy parameters for 2GB DRAM with 32nm technology. Given a fixed DRAM size, the area is dependent on the layout of those components. For multi-bank, the area increases significantly mainly because of the area of large row buffers. It should be noted that when the DRAM chip is divided into more banks, the DRAM arrays become smaller so that the CACTI can figure out a smaller area (e.g., area of 128-bank is smaller than area of 64-bank). The multi-row-buffer approach and the multi-memory-controller approach also suffer from the area penalty. In particular, multi-memory-controller has the biggest area cost because adding one memory controller means adding a set of banks as well as consuming chip pins which have been scarce resources in multicore systems. However, VCM almost has no area cost, increasing area by only 0.5% for 32 1KB-channels and 7.7% for 128 channels.

We evaluate power consumption in both high performance (HP) mode and low power standby (LPS) mode. For individual DRAM operations, experimental results show that there are no significant differences among different configurations. For 32-channel VCM in HP mode, each read operation consumes about 2.19 nJ, increasing slightly by about 3.6% compared to the baseline 8-bank DRAM.

6. Related Works

Performance Issue: Rixner et al. proposed FR-FCFS [27] scheduling algorithm which could improve memory band-width by 40% 93% for streaming applications. McKee et al. [16] showed that dynamically reordering memory requests can increase the row buffer hit rate for scientific and multimedia applications. Hur et al. [9] proposed the adaptive history-based (AHB) scheduler which improves performance by 7.6% 15.6%. There is still a lot of work [37] focusing on reordering memory requests. However, most of these studies are aim to take full advantage of the row buffer within each bank, still being limited to bank-level parallelism. Agrawal et al. [2] proposed virtually pipelined memory (VPM) which provided a deeper pipeline for handling memory requests. Although VPM might increase access individual request latency, it is able to improve effective memory bandwidth. This is because that pipelining allows more on-the-fly memory requests. Nevertheless, these schemes do not consider enhancing MLP.

VCM represents a number of approaches which add additional buffer into DRAM chip to exploit MLP. The similar idea was first proposed by Alexander and Kedem [3]. They proposed to integrate some small buffers in DRAM chip as prediction table for a DRAM prefetching scheme. Zhao et al. [34] implemented an additional off chip SRAM Cache to exploit locality for larger workload with higher bandwidth. Jiang et al. [11] further improved the DRAM Cache with some effective filters to cache only hot pages.

Several studies focus on changing DRAM’s organization in order to improve performance as well as reduce power. Zheng et al. proposed Mini-rank [35] and decoupled-DIMM to address DRAM power issue. Yamauchi et al. [32] present hierarchical multi-bank DRAM composed of 8 subbanks, improving performance about 65%. More recently, Udipi et al. [31] argued that traditional DRAM system has already not suitable for multicore system because of overfetch problem which wastes significant energy. They proposed two main aggressive schemes including Selective Bit-line Activation (SBA) and Single Subarry Access (SSA) which are able to reduce energy by 5X 6X and performance by 54% due to reduced queuing delays.

QoS Issue: Both of Rafique et al. [25] and Nesbit et al. studied QoS based on fair queuing mechanism. Mutlu et al. proposed STFM [21] and PAR-BS to solve unfairness problems for multicore system. Furthermore, Kim et al. [13] proposed ATLAS algorithm for multiple memory controllers system which could improve system throughput by 8.4% 10.8%. Eiman et al. proposed FST algorithm which throttles down cores causing unfairness by limiting the number of their available MSHRs.

7. Conclusion

In this paper, we have found that MLP has a stronger correlation with the performance of DRAM system on multicore/many-core architecture than RBHR which is considered as the only inherent metric to measure the performance of DRAM system in the past decade. In order to leverage the unexploited MLP existing in multi-core/many-core system, we have selected and evaluated four representative approaches by measuring performance, QoS, power overhead and area cost. According to the experimental results, we have found that the obsolete VCM exhibits better than the other three approaches. We argue that memory chip vendors could reconsider the VCM technology for multi-core/many-core architecture.

References

[1] CACTI: An Integrated Cache and Memory Access Time, Cycle Time, Area, Leakage, and Dynamic Power Model. http://www.hpl.hp.com/research/cacti/.

[2] B. Agrawal and T. Sherwood. Virtually Pipelined Network Memory. In: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, (**2006**), pp. 197–207.

[3] T. Alexander and G. Kedem. Distributed Prefetch-buffer/Cache Design for High Performance Memory Systems. In: Proceedings of the 2nd IEEE Symposium on High-Performance Computer Architecture, (**1996**), pp. 254–263.

[4] S. Bhansali, W.-K. Chen, S. D. Jong, A. Edwards, R. Murray, M. Drinic, D. Mihocka, and J. Chau. Framework for instruction-level tracing and analysis of program executions. In: Proceedings of the 2nd International Conference on Virtual Execution Environments, (**2006**), pp. 154–163.

[5] Y. Chou, B. Fahs, and S. Abraham. Microarchitecture Optimizations for Exploiting Memory-Level Parallelism. In: Proceedings of the 31st Annual International Symposium on Computer Architecture, (**2004**), pp. 76–87.

[6] C. Yan, Y. Zhang, F. Dai, X. Wang, L. Li and Q. Dai, "Parallel deblocking filter for HEVC on many-core processor", Electronics Letters, vol. 50, no. 5, **(2014)**, pp. 367-368.

[7] E. Ebrahimi, L. Chang Joo, M. Onur, and N. P. Yale. Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems. In: Proceedings of the fifteenth Architectural Support for Programming Languages and Operating Systems (ASPLOS), (**2010**), pp. 335–346.

[7] S. Everman and L. Eeckhout. A Memory-Level Parallelism Aware Fetch Policy for SMT Processors. In: Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture, (**2007**), pp. 240–249.

[8] A. Glew. "MLP yes! ILP no!". In: ASPLOS Wild and Crazy Idea Session, (**1998**)

[9] I. Hur and C. Lin. Memory scheduling for modern microprocessors. ACM Trans. Comput. Syst. 25, 4, Article 10.

[10] Chenggang Yan, Yongdong Zhang, Feng Dai, Jun Zhang, Liang Li and Qionghai Dai, "Efficient Parallel HEVC Intra Prediction on Many-core Processor", Electronics Letters,50(11): 805-806 (**2014**)

[11] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian. CHOP: Integrating DRAM Caches for CMP Server Platforms. IEEE Micro, vol. 31, no. 1, (**2010**), pp. 99–108.

[12] K. Luo, J. Gummaraju, and M. Franklin. Balancing throughput and fairness in SMT processors. In: IEEE International Symposium on Performance Analysis of Systems and Software, (**2001**), pp. 164–171.

[13] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter. ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers. In: Proceedings of the 16th International Symposium on High-Performance Computer Architecture (HPCA), (**2010**), pp. 1–12.

[14] C. Yan, Y. Zhang, F. Dai, X. Wang, L. Li and Q. Dai, "Parallel deblocking filter for HEVC on many-core processor", Electronics Letters, vol. 50, no. 5, (**2014)** , pp. 367-368.

[15] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In: Proceedings of the 2005 ACM SIGPLAN conference on Programming Language Design and Implementation, (**2005**), pp. 190–200

[16] S. A. Mckee, W. A. Wulf, J. H. Aylor, M. H. Salinas, R. H. Klenke, S. I. Hong, and D. A. B. Weikle. Dynamic Access Ordering for Streamed Computations. IEEE Trans. Comput., vol. 49, vol. 11, (**2000**), pp. 1255-1271

[17] Micron, 1Gb DDR2 SDRAM Component: MT47H128M8HQ-25.http://download.micron.com/ pdf/datasheets/dram/ddr2/1GbDDR2.pdf, (**2007**).

[18] Yongdong Zhang, Chenggang Yan, Feng Dai and Yike Ma, "Efficient Parallel Framework for H.264/AVC Deblocking Filter on Many-core Platform", IEEE Transactions on Multimedia, vol. 14, no. 3, (**2012**), pp. 510-524.

[19] T. Moscibroda and O. Mutlu. Memory performance attacks: denial of memory service in multi-core systems. In: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium. Article 18, (**2007**).

[20] C. Yan, Y. Zhang, F. Dai, J. Zhang, L. Li and Q. Dai, "Efficient Parallel HEVC Intra Prediction on Many-core Processor", Electronics Letters, vol. 50, no. 11, (**2014**), pp. 805-806

[21] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors", Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, (**2007**), pp. 146–160.

Copyright ⓒ 2016 SERSC

[22] Nec, 64M-bit Virtual Channel SDRAM data sheet, (**1998**).

[23] Chenggang Yan, Yongdong Zhang, Jizheng Xu, Feng Dai, Liang Li, Qionghai Dai and Feng Wu, "A Highly Parallel Framework for HEVC Coding Unit Partitioning Tree Decision on Many-core Processors", IEEE Signal Processing letters, vol. 21, no. 5, **(2014)**, pp. 573-576.

[24] M. Qureshi, D. Lynch, O. Mutlu, and Y. Patt. A Case for MLP-Aware Cache Replacement. In: Proceedings of the 33rd Annual International Symposium on Computer Architecture, (**2006**), pp. 167-178

[25] N. Rafique, W.-T. Lim, and M. Thottethodi. Effective Management of DRAM Bandwidth in Multicore Processors. In: Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques (PACT), (**2007**), pp. 245–258.

[26] S. Rixner. Memory Controller Optimizations forWeb Servers. In: Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture (**2004**), pp. 355–366

[27] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory Access Scheduling. In: Proceedings of the 27th Annual International Symposium on Computer Architecture, (**2000**), pp.128–138.

[28] C. Yan, Y. Zhang, J. Xu, F. Dai, L. Li, Q. Dai and F. Wu, "A Highly Parallel Framework for HEVC Coding Unit Partitioning Tree Decision on Many-core Processors", IEEE Signal Processing letters, vol. 21, no. 5, (**2014**), pp. 573-576.

[29] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. In: Proceedings of the ninth International Conference on Architectural Support for Programming Languages and Operating Systems (**2000**), pp. 234–244.

[30] Chenggang Yan, Yongdong Zhang, Jizheng Xu, Feng Dai, Jun Zhang, Qionghai Dai, and Feng Wu, "Efficient Parallel Framework for HEVC Motion Estimation on Many-core Processors", IEEE Transactions on Circuits and Systems for Video Technology, vol. 24, no. 12, (**2014**), pp. 2077-2089.

[31] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi. Rethinking DRAM design and organization for energy-constrained multi-cores. In: Proceedings of the 37th annual international symposium on Computer architecture, (**2010**), pp. 175–186

[32] T. Yamauchi, L. Hammond, and K. Olukotun. The Hierarchical MultiBank DRAM: A High-Performance Architecture for Memory Integrated with Processors. In: Proceedings of the 19th Conference on Advanced Research in VLSI, (**1997**) pp. 303–319.

[33] Chenggang Yan, Yongdong Zhang, Feng Dai and Liang Li, "Highly Parallel Framework for HEVC Motion Estimation on Many-core Platform", Data Compression Conference, **(2013)**.

[34] L. Zhao, R. Iyer, R. Illikkal, and D. Newell. Exploring DRAM cache architectures for CMP server platforms. In: 25th International Conference on Computer Design (ICCD), (**2007**), pp. 55–62.

[35] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In: Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture, (**2008**), pp. 210–221.

[36] C. Yan, F. Dai, Y. Zhang, Y. Ma, L. Chen, L. Fan, Y. Zheng, "Parallel Deblocking Filter for H.264/AVC implemented on Tile64 Platform", International Conference on Multimedia and Expo, (**2011**).

[37] C. Yan, Y. Zhang, J. Xu, F. Dai, J. Zhang, Q. Dai, and F. Wu, "Efficient Parallel Framework for HEVC Motion Estimation on Many-core Processors", IEEE Transactions on Circuits and Systems for Video Technology, vol. 24, no. 12, (**2014**), pp. 2077-2089.

Copyright ⓒ 2016 SERSC