Chen LC, Chen MY, Ruan Y *et al.* MIMS: Towards a message interface based memory system. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 29(2): 255–272 Mar. 2014. DOI 10.1007/s11390-014-1428-7

MIMS: Towards a Message Interface Based Memory System

Li-Cheng Chen[1,2] (陈荔城), *Student Member, CCF, ACM, IEEE*
Ming-Yu Chen[1,*] (陈明宇), *Member, CCF, ACM, IEEE*, Yuan Ruan[1] (阮　元), *Member, CCF, ACM*
Yong-Bing Huang[1,2] (黄永兵), *Student Member, CCF, ACM, IEEE*
Ze-Han Cui[1,2] (崔泽汉), *Student Member, CCF, ACM, IEEE*, Tian-Yue Lu[1,2] (卢天越)
and Yun-Gang Bao[1] (包云岗), *Member, CCF, ACM, IEEE*

[1] *State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences Beijing 100190, China*

[2] *University of Chinese Academy of Sciences, Beijing 100049, China*

E-mail: {chenlicheng, cmy, ruanyuan, huangyongbing, cuizehan, lutianyue, baoyg}@ict.ac.cn

Received November 13, 2013; revised January 14, 2014.

Abstract The decades-old synchronous memory bus interface has restricted many innovations in the memory system, which is facing various challenges (or walls) in the era of multi-core and big data. In this paper, we argue that a message-based interface should be adopted to replace the traditional bus-based interface in the memory system. A novel message interface based memory system called MIMS is proposed. The key innovation of MIMS is that processors communicate with the memory system through a universal and flexible message packet interface. Each message packet is allowed to encapsulate multiple memory requests (or commands) and additional semantic information. The memory system is more intelligent and active by equipping with a local buffer scheduler, which is responsible for processing packets, scheduling memory requests, preparing responses, and executing specific commands with the help of semantic information. Under the MIMS framework, many previous innovations on memory architecture as well as new optimization opportunities such as address compression and continuous requests combination can be naturally incorporated. The experimental results on a 16-core cycle-detailed simulation system show that: with accurate granularity message, MIMS can improve system performance by 53.21% and reduce energy delay product (EDP) by 55.90%. Furthermore, it can improve effective bandwidth utilization by 62.42% and reduce memory access latency by 51% on average.

Keywords message interface, memory system, asynchronous, granularity, semantic information

1 Introduction

The exponential growth of the number of cores (computing resource) and the amount of data needs to be processed (working set) places heavy pressure on main memory system for high throughput, high bandwidth, high parallelism, large capacity, etc. The number of cores integrated into a processor chip is expected to double every 18 months①, and some many-core processors have already been on the market, such as 60-core Intel® Xeon Phi™ coprocessor②, 72-core Tilera TILE-Gx 64-bit processor③. The increasing number of cores demands for higher memory bandwidth and higher memory parallelism. Furthermore, memory requests from different cores tend to interfere with each other, which will result in poor memory locality (e.g., low row buffer hit rate). Thus how to leverage parallelism is considered to have a higher priority than locality in future memory architectures[1]. On the other hand, in the era of big data, the amount of data needs to be

Regular Paper
This work is partially supported by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06010401, the National Basic Research 973 Program of China under Grant No. 2011CB302502, the National Natural Science Foundation of China under Grant Nos. 60925009, 61221062, 61331008, and the Huawei Research Program under Grant No. YBCB2011030.

*Corresponding Author

①International technology roadmap for semiconductors. http://www.itrs.net/Links/2011ITRS/Home2011.htm, Dec. 2013.

②Intel® Xeon Phi™ Coprocessor: Datasheet. https://www-ssl.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-phi-coprocessor-datasheet.pdf, Dec. 2013.

③TILE-Gx Processor Family. http://www.tilera.com/products/processors/TILE-Gx_Family, Dec. 2013.

processed is predicted to grow with a rate of 40% per year[④]. Big data processing requires larger memory capacity to put more data in-memory and higher memory bandwidth to achieve better performance.

However the Synchronous Dynamic Random Access Memory (SDRAM) based memory interface (e.g., DDRx), which acts as the bridge between low-level cores and high-level data, fails to scale, and it has been considered as the major bottleneck in a chip multiprocessor (CMP) system. Besides the well-known memory wall problem[2], it also faces many other challenges (or walls), which are concluded as follows:

Memory Wall (*Latency*). The original "memory wall" refers to memory access latency problem[2], which was a major problem in the memory system until mid-2000s. Then came the multi-core/many-core era after the processor frequency race slowed down. The situation has changed a bit that queuing delay at the memory controller has become a major bottleneck, and it can contribute up to 70% of the total memory latency in an 8-core system[1]. Thus for future memory architecture, it shall place a higher priority on reducing queuing delay. Exploiting higher parallelism in memory system can effectively reduce queuing delay because it is able to de-queue requests faster[1]. However the synchronous design of traditional DDRx memory has natural limitation on leveraging parallelism.

Bandwidth Wall. The increasing number of concurrent memory requests and the increasing size of working set will place heavy pressure on memory bandwidth. However since the relatively slow growth of pin counts of processor module (about 10% per year), the bandwidth of memory system fails to scale with the number of cores. Actually, the average memory bandwidth for each core is predicted to be decreasing, which has been concluded as bandwidth wall[3]. In traditional DDRx memory system, the memory controller (usually integrated into processor chip) connects directly with DRAM memory channels through synchronous parallel bus, which costs a large number of processor pins (e.g., 240 processor pins for a DDR3 DRAM channel). Adopting faster and narrower serial link bus (communicates between cores and memory channels) can effectively alleviate the processor pin limitation, such as fully-buffered DIMM, Intel® Scalable Memory Interconnect, buffer-on-board memory. Further optimizations such as optical interconnection and 3D stacking are supposed to solve bandwidth problem substantially.

Efficiency Problem. Latency and bandwidth are two of the physical factors of a memory system while the efficiency of memory access really counts for processor. In traditional DDRx DRAM memory system, the memory controller always reads or writes a cache-block data (e.g., 64 B) within a burst-length (BL) burst (BL is 8 in DDR3) in spite of which word is really needed. Before that, a whole row data (e.g., 8 KB) needs to be activated and stored in the row buffer within the destination memory-rank. This fixed-size and coarse-grained design can improve memory bandwidth when memory accesses have good locality. However, in multi-core or many-core systems, the locality both in cache block and row buffer is lost due to memory requests interference among cores[1,4]. It has been shown that coarse-grained data cache has almost no performance benefit for some scale-out workloads[5-6]. The well-known over-fetch problem, that most of the read-in data have never been used by the processor, will be amplified in such situation. Thus for memory accesses with low spatial locality, coarse-grained data unit will waste much memory bandwidth and power. Memory systems that support fine granularity access[4,7] can improve both the bandwidth and power efficiency.

Capacity Wall. Big data processing workloads require larger memory capacity to put more data in-memory. However the capacity of DDRx DRAM memory fails to scale with the demand. The number of memory channels supported by a processor is limited by the available pin counts. And the number of DIMMs (or ranks) which can be supported within each DDRx channel is limited by signal integrity restriction. For instance, the maximum number of DIMMs supported in each DDR1 channel can be four, while it is reduced to only one in each DDR3-1600MHz channel[8]. Furthermore, the capacity provided by each DIMM is growing slowly due to the difficulties in decreasing the size of capacitor in DRAM cell[9]. To increase the number of DIMMs supported in each channel, registers and buffers have been added into the memory module to improve signal integrity, such as RDIMM and LRDIMM memory. Furthermore, there are also various proposals to provide big-capacity memory, such as buffer-on-board (BOB) memory[9], Hybrid Memory Cube (HMC) memory[⑤], high density non-volatile memory (e.g., PCM)[10] and 3D-stacked memory[11].

Power Wall. It has been reported that memory power can contribute about 40% to the total system power in large servers[12-13]. Leaking in capacitor-based DRAM cells contributes the major static power

[④]Oracle: Big data for the enterprise. http://www.oracle.com/us/products/database/big-data-for-enterprise-519135.pdf, Dec. 2013.

[⑤]Hybrid Memory Cube Consortium. Hybrid memory cube specification 1.0. http://www.hybridmemorycube.org/files/HMC_Specification%201_0.pdf, Dec. 2013.

of DRAM system (including periodic refresh power), which is not an architectural issue. However, due to the low locality problem mentioned above, a large portion of dynamic power in coarse-grained DRAM memory is actually wasted on activating and reading/writing useless data. Changing DDRx memory systems to support sub-access in row buffer and fine granularity memory access can alleviate the over-fetch problem. To reduce DRAM static power, non-volatile memory (e.g., PCM) can be investigated as potential alternatives for existing memory technologies. However non-volatile memory (NVM) usually has totally different access parameters and timing constraints, so it cannot work under a traditional synchronous memory interface designed for DRAM such as DDRx.

Reliability Problem. Memory reliability is increasingly a concern with the rapid improvement of memory density and capacity. Error correction codes (ECCs) are the dominant solution for nowadays server memory. However it has been shown that single error correction, double error detection (SECDED) is insufficient for future memory system[14]. There are other stronger memory protection schemes such as chip-kill which is not widely adopted. In traditional DDRx memory system, a fixed hardware error correction mechanism is adopted and dedicated data bus pins are assigned to ECC. The granularity and code pattern are all fixed and processed by hardware logic (in integrated memory controller). Recently many studies have contributed to investigating more flexible error protection approaches, especially when adopting sub-access memory and considering to introduce non-volatile memory into memory systems[15].

Besides all the above walls, there is a long research trend to equip the memory system with some simple processing logic to make memory more autonomous. Logic in memory can significantly reduce data transfer between the memory system and processor (because data can be processed in local), thus it can improve performance and reduce power consumption. Many autonomous memory systems have been proposed, such as Processing in Memory (PIM)[16], Active Memory Operation (AMO)[17-18], and Huawei Smart Memory[19]. However these technologies are limited to proprietary designs and never get into the standard DDRx memory interface, which supports read and write operations only.

In summary, a large body of previous work has been contributed to alleviating various memory bottlenecks (or walls). However each of them only focuses on one or part of these walls. To the best of our knowledge, none of them is able to provide a universal solution to all these problems. Table 1 gives a summary of these approaches and the problems they address (please refer to Section 2 for details). For example, the BOB[9] memory system focuses on addressing DRAM memory bandwidth and capacity wall, in which a simpler controller is placed on board to receive memory requests (in packets) and schedule memory requests to DRAM devices, thus it makes the BOB memory system a little autonomous.

Table 1. Comparison of Different Approaches to Alleviating Various Memory Walls

	LY	BW	CY	EY	PR	AS
Sub-Access	$*$	×	×	$*$	√	×
FGMS	$*$	×	×	√	√	×
Buffer-Chip	×	×	√	×	×	×
BOB MS	×	√	√	×	×	$*$
AMO	×	×	×	×	√	√
NVM	×	×	√	×	√	$*$
3D-Stacked	×	$*$	√	×	√	$*$
Photonics	×	√	√	$*$	×	$*$
Mobile	×	×	$*$	×	√	$*$

Note: √: yes, ×: no, $*$: maybe, LY: latency, BW: bandwidth, CY: capacity, EY: efficiency, PR: power, AS: autonomous.

In this work, we argue that traditional synchronous bus-based memory interface should be redesigned to incorporate more future innovations. In contrast to traditional read or write bus transaction based memory interface, a flexible asynchronous message based memory interface will bring more design opportunities. We propose a universal message interface based memory system called MIMS. In MIMS, memory requests and responses are transferred over faster and narrower serial link bus with high-level message protocol. A local buffer scheduler is placed between the on-chip memory controller and DRAM memory modules (or channels). Device-specific scheduling and timing constraints are decoupled from the on-chip memory controller, which is simplified to only encapsulate memory requests into packets and process responses. The memory controller communicates with the buffer scheduler over high-speed serial bus with a flexible message packet protocol. Each message packet is allowed to encapsulate multiple memory requests or responses and additional semantic information such as granularity, thread ID, priority, and timeout. The buffer scheduler acts as the traditional memory controller: it needs to track the status of local memory devices, schedule memory requests, generate and issue specific local memory bus commands, meanwhile fulfilling the timing constraints. Additionally, the buffer scheduler can leverage semantic information (from the processor) to optimize memory scheduling.

MIMS will bring at least the following advantages:

1) It provides a universal, scalable message interface to access different memory subsystems. The status tracking and request scheduling are decoupled from the

memory controller and placed down on the buffer scheduler. Thus the integrated memory controller has no timing limitations and it is adaptive to work with other emerging memory technologies. What is more, the memory capacity is only restricted by the buffer scheduler, which is decoupled from the on-chip memory controller (processor pin).

2) It can naturally support variable granularity memory requests. Each memory request is transferred with the exact size of really useful data. This can significantly improve data/bandwidth efficiency and reduce memory power consumption.

3) It enables inter-request optimizations during the memory access, such as encapsulating multiple memory requests in a message packet (when they access the same destination memory module), compressing memory addresses of a sequence of requests, and combining contiguous requests.

4) It is easy to add additional semantic information in a message packet to help optimize memory requests scheduling in the buffer scheduler. Local computation or intelligent memory operation requests can also be added as part of the message.

5) It enables independent design of the buffer scheduler and memory modules. The memory system can be upgraded after the CPU module has been released (which is impossible for integrated DDRx memory controller). One type of processor can connect to multiple memory-side designs with different organization, redundancy, memory chips, and acceleration logics.

To demonstrate the benefits of using a message interface in memory system, we have implemented a cycle-detailed memory system simulator called MIMSim. Trace-driven experiments are performed for multithread workloads with fine-grained memory access. The results provide elementary proof for the benefits of MIMS on performance, effective bandwidth utilization and power reduction. It also answers a basic question, whether the additional latency for message processing will effect on the overall performance.

The rest of the paper is organized as follows. Section 2 gives an overview of the memory system and related work, while Section 3 presents the Message Interface Based Memory System (MIMS), includes its architecture, packet format, packet decoding and some challenges. Section 4 presents the experimental setup, and the results and discussions are presented in Section 5. Section 6 concludes this paper.

2 Background and Related Work

In this section, we first give a brief description of dominant JEDEC-style DDRx SDRAM memory system, then discuss some optimizations on memory architecture, including sub-access memory, memory reliability, buffer-chip memory, autonomous memory, and some aggressive memory systems, such as non-volatile memory (NVM), 3D-stacked memory, and photonics interconnect memory.

2.1 DDRx Memory System

The JEDEC standard DDR[6] (Double Data-Rate) synchronous DRAM is dominant nowadays. In contemporary DDRx memory system, the memory controller is usually integrated into processor chip, and serves as the bridge logic between the processor core and off-chip DRAM devices (or chips). The memory controller is responsible for receiving memory requests from the last level cache (cache miss) and scheduling memory requests to DRAM chips. The memory controller needs to track the status of DRAM devices (e.g., bank states) and generate DRAM commands for selected requests. The scheduling needs to meet the DDRx timing constraints. The integrated memory controller communicates with DRAM devices over parallel synchronous DDR bus (with separate data, command, and address bus). This wide synchronous bus design results in high processor pin-count overhead, and it has become a bottleneck to support large capacity memory, since the processor pin is an expensive resource and the growth of processor pin-count fails to keep up with the demand.

The DDRx memory system has a hierarchical organization, with available parallelism at each level. Each memory controller can support multiple memory channels, and each channel has dedicated DDRx bus, thus each channel can be accessed independently. Within a memory channel, there might be multiple DIMMs. Each DIMM might consist of multiple ranks (e.g., 1, 2, 4), and each rank provides a logical 64-bit data-path (bus) to the memory controller (it is 72-bit for ECC-DIMM). Multiple DRAM devices within a rank need to be operated in tandem. DRAM device with 8-bit data width (x8) is the most commonly used DRAM device today, and it will be referred in this work by default.

A DRAM device consists of multiple DRAM banks (it is 8 in DDR3) and these banks can be accessed concurrently. Within each DRAM bank, there is a two-dimensional (2D) data array, consisting of rows and columns. A row buffer is dedicated to each bank, which is usual 4 KB~16 KB. Before each column access, a whole destination row data needs to be activated into the row buffer by an *Active* command. If following requests access the same row opened in the row buffer (it is a row buffer hit), it can be accessed directly with column read/write commands, which have shorter la-

[6]DDR3 SDRAM standard, http://www.jedec.org/standards-documents/docs/jesd-79-3d, Dec. 2013.

tency. Otherwise, the data in the row buffer needs to be *Precharged* back into DRAM array before issuing new requests (it is a row buffer conflict).

2.2 Sub-Access Memory

Sub-access memory refers to dividing a whole DRAM component into multiple sub-components, so that each memory request only needs to access data in a sub-component. Here DRAM component can be referred to rank, row buffer and cache block. Sub-access memory can reduce memory access power and improve memory level parallelism (MLP).

Sub-ranked memory system divides a 64-bit memory rank into multiple narrower logical sub-ranks. Each sub-rank has fewer DRAM devices and narrower data bus, and they can be accessed concurrently. Data layout needs to be adjusted so that each cache block is placed into a single sub-rank. Each memory request only requires a part of memory devices (in the same sub-rank) to be activated and accessed. Thus sub-rank technology can effectively reduce memory power by alleviating the over-fetch problem and improve MLP. The downside of it is that the access latency of each memory request will increase since only part of the total memory bandwidth can be utilized, if they still adopt fixed coarse granularity memory access.

Rambus's module-threading technique[20] divides a memory channel into multiple thread modules, and each thread module is selected by a separate chip select signal and connected with independent narrower data bus, thus they can be accessed independently. Supporting fine granularity memory access, the module-threading technique can improve memory throughput and parallelism and reduce power consumption.

Multicore DIMM (MCDIMM)[21-22] was proposed to save dynamic power by reducing memory over-fetch. It achieves this by dividing a memory rank into multiple narrower data-path subsets, and only one subset needs to be accessed for each memory request rather than a whole rank. Thus data transfer takes longer for coarse-grained memory access. A demux register is placed on each rank, which routes control signals to the destination rank subset. Since the data bus is split into multiple independent subsets, it might encounter data load-unbalance problem if memory accesses are not evenly distributed among subsets (partial data bus can be unutilized).

Zheng *et al.* proposed Mini-Rank[23] to improve memory power efficiency. A small bridge chip named MRB (mini-rank buffer) is added in each DRAM DIMM between DRAM devices and DDRx bus, which breaks each 64-bit rank into multiple narrower mini-ranks. The MRB is responsible for producing chip select signals to the destination mini-rank based on memory address, and it also needs to relay data transfer between internal narrower mini-rank data-path and external wider DDRx bus. It is worth noting that, the outer DDRx data bus is not split into multiple mini-ranks, and data is always transferred through the full 64-bit DDRx bus, which prevents it from the load-unbalance problem. Furthermore, Fang *et al.* proposed heterogeneous mini-rank[24] to support near-optimal configuration for each workload based on its memory access behavior and its memory bandwidth requirement. A memory type mapping table (MEMTMT) is added into the memory controller to store type information of workloads. The memory controller needs to first check the mini-rank type for each memory request.

Zhang *et al.* proposed heterogeneous multi-channel (HMC)[25] to balance the performance and power consumption of the DRAM system. HMC groups physical DRAM devices into multiple logical sub-ranks with different width of data bus, and these heterogeneous sub-ranks can be controlled and accessed simultaneously. They also proposed a novel memory access scheduling algorithm to improve data bus parallelism and achieve high DRAM data bus utilization by grouping together memory requests with the same channel (data width).

Yoon *et al.* proposed adaptive granularity memory systems (AGMS)[4] to combine the best of fine-grained and coarse-grained memory accesses. They adopted sub-ranked memory systems to implement adaptive granularity. They augmented the virtual memory interface to allow software to specify the preferred access granularity for each page, and this information is stored in a page table entry when the page is allocated. The access granularity can be determined by spatial locality: coarse-grained data accesses are used for high spatial locality applications and fine-grained data accesses are used for low spatial locality applications. They further proposed the dynamic granularity memory system (DGMS)[7] to dynamically adapt memory access granularity without requiring software or OS support. A two-level hardware prediction control mechanism was proposed: a local prediction controller in each core and a global prediction controller at the memory controller. The access granularity of a memory request is determined based on its accessing history.

Udipi *et al.* proposed Selective Bit-Line Activation (SBA) and Single Subarray Access (SSA)[1] to address over-fetch problem and thus reduce memory power. In SBA, each memory access just activates exactly those bitlines that provide the requested cache line instead of a whole row data. In SSA, an entire cache line is accessed in a single subarray by re-organizing the data layout in SSA. Cooper-Balis and Jacob[26] proposed a fine-grained activation approach to reduce memory active power, which only activates a smaller portion of a row within the data array by utilizing posted-CAS command. Convey-designed Scatter-Gather DIMMs[27] are designed to support 8 bytes (fine granularity) access that can reduce the inefficiency in non-unity strides or randomly memory accesses, however the implementation detail of SGDIMM is lack. Cray Black Widow[28] adopts many 32-bit wide channels, allowing it to support minimum 16-byte memory access granularity.

2.3 Reliability for Fine Granularity Memory

Memory reliability becomes a concern with the rapid improvement of memory density and capacity. However, it is not straightforward to provide reliability for fine granularity memory, which might induce heavy overhead. For example in multi-core DIMM[21], each sub rank needs to be protected by chip-kill mechanism independently, which will result in high storage overhead (up to 37.5%).

Zheng *et al.*[23] briefly discussed the design options to support ECC in the mini-rank architecture. They suggested two approaches: the first one is to distribute ECC-bit blocks over all devices as in RAID-5, thus a mini-rank is no longer composed of a fixed set of devices and each access needs to touch one more device than the simple mini-rank design. The second one is called embedded ECC, which proposes to store ECC bits along with their associated data bits in the same page, thus it needs to change the memory data layout. The main difficulty is how to perform address translation, which can be solved as in the prime memory system. Furthermore, the checking function (or logic) can be implemented in the mini-rank buffers to avoid extra traffic on memory bus.

As shown in [4], directly supporting ECC for fine-grained memory access with sub-ranked memory is expensive, since each sub-rank requires at least one extra DRAM chip for ECC, and the overhead of ECC for a 8-byte granularity memory access is up to 100% (8-byte data + 8-byte ECC). To reduce the ECC overhead, the authors proposed a unified data/ECC layout for both coarse-grained and fine-grained accesses simultaneously[7]. They spread ECC blocks across sub-ranks in a uniform, deterministic fashion similar to RAID-5. To further tolerate a pin failure, they proposed a chipkill-level protection for DGMS, which added a 7-bit CRC error detection for each 16B data block, and extra parity block is stored in an extra DRAM chip which is used to correct the error.

Udipi *et al.*[1] proposed an RAID-5 approach to provide chip-kill protection for the SSA (single subarray access) DRAM architecture. In SSA design, a cache line and its associated checksum are placed in a single subarray within a single DRAM chip, and the checksum verification is done for each memory access in the memory controller. An extra DRAM chip is added in the DIMM, and every eight cache lines have an associated global parity, and the global parity is evenly distributed into all the nine DRAM chips in the DIMM.

Enhanced embedded ECC[29] suggests to store data bits and the associated ECC bits in the same DRAM device and in the same DRAM row, thus accessing the associated ECC bits just requires another column access. The extra ECC access is always row buffer hit to reduce the overhead. The authors of [29] also proposed a novel address-mapping scheme called Biased Chinese Reminder Mapping (BCRM) to implement page interleaving on E3CC without using expensive integer division. To further reduce the ECC traffic, they added an ECC cache in the memory controller to buffer and exploit the locality of the ECC bits.

Another efficient approach to providing high memory reliability meanwhile maintaining low overhead is to use multi-tiered error protection technique that decouples the process of error detection from error correction. The drawback of it is that it needs one or more extra memory accesses to get error correction information when DRAM error happens. Virtualized ECC[30] detaches the physical mapping of data from the physical mapping of its associated ECC information; error detection information is stored in extra chips of the ECC DIMM while the full redundant information required for correction is mapped in data chips. The authors of [30] further suggested to use processor cache for ECC information to improve ECC access bandwidth. Li *et al.*[14] proposed to use a staged Bose Chaudhuri-Hocquenghem (BCH) code for exascale memory systems. They divided the complex error recovery procedure into three ECC logic paths to provide low ECC latency and high error coverage. Chen *et al.*[29] also suggested adopting BCH codes to reduce storage overhead in E3CC. Udipi *et al.*[31] proposed a localized and multi-tiered protection scheme named LOT-ECC, which employs simple checksum and parity codes. They separated error detection and error correction functionality. Data and codes are localized to the same DRAM row to improve memory access efficiency. Yoon *et al.*[15] also proposed FREE-p for non-volatile memory which adopts multi-tiered BCH code.

2.4 Buffer-Chip Memory

To alleviate memory capacity problem, a common way is to put an intermediate buffer (logic) between the memory controller and DRAM devices, which can reduce the electrical load on the memory controller and improve signal integrity.

In registered DIMM (RDIMM)⑦, a simple register is integrated on DIMM module to buffer control and address signals. Load reduced DIMM (LRDIMM)⑧ further buffers all signals issued to DRAM devices, including all data and strobes. Decoupled DIMM[32] adopts a synchronization buffer to convert signals between low speed memory devices and high data rate memory bus. With a similar idea, BOOM[33] adds a buffer chip between the fast DDR3 memory bus and wide internal bus, which enables the use of low-frequency mobile DRAM devices, thus BOOM could efficiently reduce memory power.

In fully-buffered DIMM (FBDIMM)⑨ memory module, there is an AMB (advanced memory buffer) integrated on each DIMM module, and multiple FB-DIMMs are organized as a daisy chain which can support large capacity. The memory controller communicates with the AMB through point-to-point narrow, high-speed channels with some simple packet protocol. Intel® Scalable Memory Interconnect (SMI)⑩ and IBM POWER7™ memory system[34-35] also place logic buffers between the processor and DRAM channels, which can also support more memory channels.

Cooper-Balis *et al.*[9] proposed a generalized buffer-on-board (BOB) memory system. In BOB, intermediate logic is placed (on motherboard) between the on-chip memory controller and DRAM devices. The memory controller communicates with the intermediate buffer through serial link bus. The memory controller is decoupled from scheduling, and the intermediate buffer actually acts as a traditional memory controller: it tracks status of its local memory devices and schedules memory requests, issues corresponding DRAM commands meanwhile meets the timing constraints. The BOB memory system is able to alleviate the capacity and bandwidth problem.

UniMA[36] aims to enable universal interoperability between processors and memory modules. Each memory module is equipped with a unified DIMM interface chip (UDIC). The memory controller sends read/write requests to UDIC through the unified interface without taking care of any device status or timing constraints.

Ham *et al.*[37] proposed disintegrated memory controller to support heterogeneous command protocols in a modular manner. The memory controller is split into two parts: master and slave. The master controller is integrated into the processor chip. It receives memory requests from the processor and then forwards them to destination salve memory controllers (based on memory address). The slave controller receives memory requests from the master and follows a technology-specific protocol when scheduling commands to memory devices meanwhile fulfilling protocol and timing constraints. The master and slaver controllers communicate over serial, point-to-point link bus for higher off-chip bandwidth with packets (encapsulating memory commands, addresses, and data).

It is worth noting that all buffer-chip memory systems listed above adopt the simple design that a memory request is dedicated to a simple packet without any semantic information, thus the packet is just a transform of traditional parallel bus transaction. Besides memory request as payload, the extra packet head and tail will introduce packet overhead.

2.5 Autonomous Memory

There has been a long time effort to make memory autonomous by equipping main memory with processing logic to support some local computations in memory. Processor-in-memory (PIM) systems incorporate processing units on modified DRAM chips[16,38]. PIMs exploit the high memory bandwidth and low latency available inside the DRAM package. Smart memory[19] makes memory system smart by keeping computation and lock management close to data in-memory, which can reduce chip I/O bandwidth, reduce latency, and achieve high performance. Fang *et al.*[17-18] proposed an intelligent memory controller that can execute active memory operations (AMOs) in memory module. They added a new hardware component called active memory unit (AMU) into each memory controller, so that operations (which are issued by processors) can be executed where the data reside. AMUs support both scalar operations that operate atomically on individual words of data and stream operations that operate on sets of words separated by a fixed stride length. AMOs can eliminate cache misses, reduce cache pollution, and reduce network traffic.

2.6 Aggressive Memory System

Non-volatile memory (such as PCM)[10,39-41] has been considered as a potential replacement for DRAM chip in the future. NVM device can eliminate static power consumption and promise to provide higher capacity. Recent results show that some NVM has comparable latency to the DRAM. However, NVM chip usually has different timing constraints so that it cannot be

⑦http://download.micron.com/pdf/datasheets/modules/ddr3/JSZF36C512x72PD.pdf, Dec. 2013.

⑧http://www.micron.com/∼/media/Documents/Products/Data%20Sheet/Modules/LRDIMM/kszf36c1g_2gx72ldz.pdf, Dec. 2013.

⑨http://download.micron.com/pdf/datasheets/modules/ddr2/HTF9C32_64_128x72F.pdf, Dec. 2013.

⑩http://www.intel.la/content/dam/doc/datasheet/7500-7510-7512-scalable-memory-buffer-datasheet.pdf, Dec. 2013.

incorporated into the synchronous DDRx memory interface directly.

On-chip optical interconnection is expected to provide high bandwidth for memory system. Udipi *et al.*[11] proposed a novel memory architecture that adopts photonics interconnects among memory controller and 3D memory stacked dies. They also proposed a novel packet based interface to relinquish the memory controller and make memory modules more autonomous. Each packet only contains a single memory request, and memory requests need to be processed in FCFS (First-Come, First-Served) order.

Hybrid Memory Cube (HMC)[11] utilizes 3D interconnect technology. A small logic layer is placed below vertical stacks of DRAM dies, and the logic layer and 3D DRAM dies are connected with through-silicon via (TSV) bonds. The logic layer is responsible for controlling and scheduling memory requests. The memory controller communicates with the HMC logic layer through abstracted high-speed interface. The logic layer is flexible to allow HMC cubes to be designed for multiple platforms and applications without changing the high-volume DRAM. All in-band communication across link is packetized in HMC, and each packet is dedicated to a single operation. Furthermore, HMC supports variable memory access granularity (16 B$\sim$128 B).

3 Message Interface Based Memory System

3.1 Why Message-Based Interface

The current synchronous bus-based memory interface can be dated back to the 1970s when the first DRAM chip in the world was produced. After 40 years the main characteristics remains unchanged: separate data, address and control signals; fixed transfer size and memory access timing (latency); CPU being aware and taking care of every bit of storage on memory devices; limited on-going operations on the interface.

One may argue that a simple and raw interface for DRAM keeps the minimum latency for memory access, but it also brings obstacles to improve memory performance as described in Section 1. Nowadays with more and more parallelism in multi-core/many-core systems, single memory access latency is not the main issue for overall performance any more. *Is it the right time to change this decades-old interface?*

Decoupling is the common trend of a lot of previous work mentioned in Section 2. That is to separate the data transfer with data organization. The CPU should only take care of sending requests and receiving responses while the buffer scheduler takes charge of local DRAM chip organization and scheduling. A packet-based interface will enable this separation by encapsulating data, address and control signals in packets. But if we just stop here, then packet is only a low-level encapsulation for bus transaction.

We can go a further step from packet to message. Here message means that the content of a packet is not predefined or fixed but programmable. Message also means the CPU can put more semantic information on a packet in addition to read/write operations. Then the buffer controller can make use of this information to achieve better performance. The information maybe request granularity, sequence, priority, process ID, or even array, link pointer and locks. It is like that the buffer controller is integrated with the processor virtually to get all necessary semantic information for memory scheduling optimization.

A message-based interface will provide many opportunities to help solve memory system issues.

For the latency problem, though a message interface might increase the latency of a single memory request, it helps improve parallelism and perform better prefetching and scheduling with the help of semantic information. This can effectively reduce queuing delay and thus decrease the overall memory access latency.

For the bandwidth problem, the message interface will support better memory parallelism to fully utilize the bandwidth; the packet interface enables new interconnection technologies. Message also enables effective address compression.

For the efficiency problem, exact data granularity information will help reduce the waste of data over-fetching; message also enables accurate prefetching to reduce unnecessary operations.

For the capacity problem, decoupling enables special design for large capacity memory systems; message even enables a network of extended memory systems.

For the power problem, message enables fine-grained control of active DRAM regions. Decoupling also enables low power NVM to be included in the memory system transparently.

For the reliability problem, decoupling enables independent design of memory organization and error correction pattern. There is no need to predefine memory region as data and ECC. With semantic information, different memory locations might have different protection levels at different time periods.

For autonomous operations, message provides a natural support with semantic information and command extension.

To demonstrate the benefits of message interface, a draft architecture design is given in the following sub-

[11] Hybrid memory cube specification 1.0. http://www.hybridmemorycube.org/files/HMC_Specification%201_0.pdf, Dec. 2013.

section. It should be noted that the design is elementary and incomplete to cover all the advantages of message-based interface.

3.2 MIMS Architecture

Fig.1 shows the architecture of the Message Interface Based Memory System (MIMS). As in the buffer-on-board (BOB) memory system[9], the memory controller which is integrated into the processor does not directly communicate with memory devices (DRAM). Instead, it communicates with the buffer scheduler over serialized point-to-point link bus which is narrower and can work at a much higher frequency. Each memory controller can support multiple buffer schedulers, and each buffer scheduler can support multiple local memory modules. The buffer scheduler consists of memory request buffer, packet generator, packet decoder, return buffer and link bus, etc.

The memory controller receives variable-granularity memory requests from multiple processor cores. The memory controller firstly selects the destination buffer scheduler based on the address mapping scheme, then the memory request is placed into on-chip network (NOC). The NOC routes each memory request to its destination request buffer over link bus. For each buffer scheduler, the request buffer is divided into read queue and write queue, which is used to buffer read and write requests respectively. By default, read requests have higher priority when scheduling requests to encapsulate into packets. If the number of write requests in the write queue exceeds the high water mark[42-43], then write requests get higher priority. Write requests will contiguously be selected to be packed and sent to the destination buffer scheduler until the number of write requests returns below the low water mark.

The packet generator is responsible for selecting multiple memory requests to encapsulate into a packet. It needs to construct packet head, which contains the meta data of a packet. Then the packet is sent to the SerDes (serialize-deserialize) buffer. Note that, the packing operation is not in the critical path, because the packet generator keeps tracking the status of the serialized link bus, and it can start the packing process in advance before the link bus becomes available (free). After the packet has been constructed and the link bus becomes available, the packet will be sent to the destination buffer scheduler over link bus.

On receiving a message packet from the on-chip memory controller, the packet decoder within the buffer scheduler will unpack the packet and retrieve all memory requests from the packet. Then these memory requests are sent to the scheduler. The scheduler acts as a traditional memory controller: it communicates with the DRAM memory module through relatively slow parallel bus with the synchronous DDRx protocol as in the traditional memory system. The scheduler needs to track all the memory module states (e.g., bank state, bus state) attached to it, and it also needs to schedule memory requests, generate DRAM commands (e.g., ACTIVE, PRECHARGE, READ, WRITE) based on the memory states, and issue DRAM commands to the memory modules meanwhile fulfilling the timing constraints.

We adopt sub-rank memory system to support variable-granularity memory access, which is also used in DGMS[7]. We adopt x8 DRAM devices, and each traditional 64-bit rank is separated into 8 sub-ranks, with one DRAM device per sub-rank, thus the width of a sub-rank is 8-bit. The data burst length (BL) is 8 as in DDR3, and the minimum granularity of a memory request is 8-byte.

3.2.1 *Packet Format*

The message packet is the essential and critical component in MIMS, and a packet should be designed to easily scale to support various memory optimizations. Each packet can support to encapsulate multiple memory requests. Furthermore, each packet needs to contain some link overhead (LKOH) which is generated and processed at lower layer such as link layer and physical

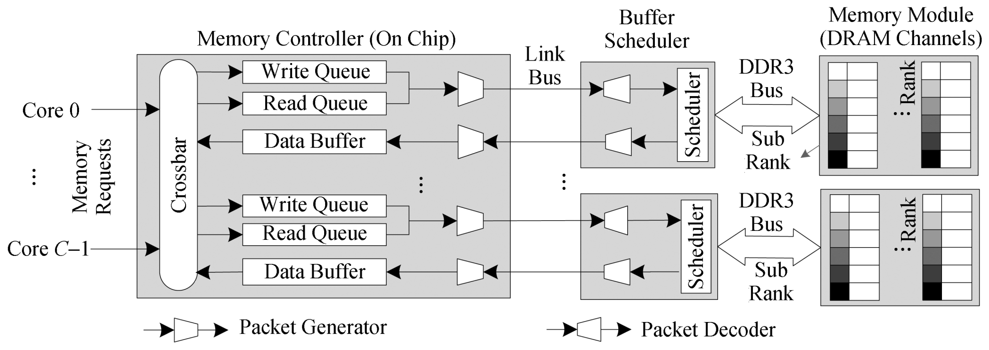


Fig.1. Message interface based memory system architecture.

layer. The LKOH is necessary for serial bus communication protocol, which usually contains reliability-related data such as start signal, end signal, sequence ID, and checksum codes (CRC).

The overhead of LKOH is high if each packet only contains a small amount of data (payload). Especially for a read request, which only contains address and operation, the overhead of LKOH would be up to 50% for a size of 8B LKOH (as in the PCIe protocol). Thus encapsulating multiple memory requests into a packet will increase the size of payload to reduce the link and packet overhead.

To encapsulate multiple memory requests in a packet, we propose a variable-length packet format for MIMS. The packet has three basic types: read packet, write packet, and read-return packet, which can contain multiple read memory requests, write requests and return data, respectively. Since a packet may contain variable number of requests, it needs a packet head (PKHD) which contains meta data of the packet. The detailed format of a read packet is shown in Fig.2. We can see that a read packet has a packet head and multiple request messages (RTMSGs), and it also contains an LKOH. The packet head contains the destination buffer scheduler identifier (DESID), packet type (PT, such as read), the count (CNT) of requests and some other reserved (RSV) fields. Note that all the requests in a packet are sent to the same destination buffer scheduler. After the packet head, multiple request messages are closely aligned. Each request message (RTMSG) represents a memory request with some additional semantic information. It basically contains address (ADDR) and granularity (GY) for each memory request, and can easily scale to contain more semantic information such as request timeout (TO), thread ID (TID). The timeout represents the longest acceptable latency (queue delay) that it can tolerate, which is valuable to implement QoS (quality of service) for memory requests. Other messages that are valuable for scheduling can also be integrated in RTMSG. All the RTMSGs in a packet are in the same format and with same length, thus encoding and decoding read packets can be direct and effective with neglected overhead.

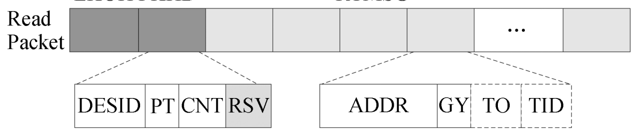


Fig.2. Read packet format.

For a write packet, as shown in Fig.3, the format is nearly the same as a read packet. It also contains an LKOH, a packet head and multiple write requests, where the packet head is the same as in a read packet, except that the packet type (PT) is write packet. Besides an RTMSG, each write request also contains write data (WTDA). The RTMSG is the same with that in a read request. Write data might be variable-length, and the length is determined by the granularity (in RTMSG) of a request. For example, the length of data is 8-byte for a fine granularity write, and it is 64-byte for a coarse granularity write.

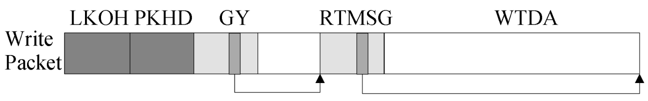


Fig.3. Write packet format.

The read-return packet has the same format with write packet. The request address needs to be returned since memory requests are scheduled in an out-of-order manner both in the on-chip memory controller (packet encoding) and in the buffer scheduler (request scheduling). In order to reduce the overhead of returning address, each read request can be assigned a request ID, which is much smaller (e.g., 10-bit for 1 024 requests).

3.2.2 Packet Decoding

Once the buffer scheduler receives a packet, the packet decoder firstly reads the packet head and gets the meta data of the packet, such as DESID (destination buffer scheduler ID), packet type, memory requests count and other reserved data. The DESID is used to check whether the packet is routed correctly. The type of the packet and the count of memory requests are checked. Then the packet can be decoded based on the packet type.

Read Packet. Since each read request message has fixed-length fields, including address, granularity, etc., it can be processed in parallel easily. Fig.4 shows the process of parallel decoding for a read packet. In this example, 4 RTMSGs are decoded in parallel. Since the format for each RTMSG is the same, they can be decoded with a single mask, and the address, granularity and other messages of each read request can be extra-

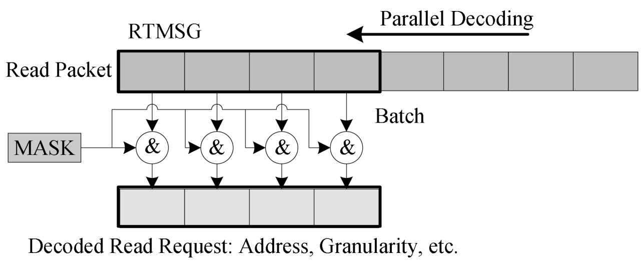


Fig.4. Parallel decoding for a read packet. Each batch (e.g., 4) of RTMSGs is decoded in parallel with MASK operator.

cted. After that, the next batch RTMSGs are ready to be decoded.

Write Packet. Each write request has the request message along with variable length of write-data, where the length can be determined by the granularity of the memory request. For example, if the granularity is 4, then the length of data is 32-byte (8-byte $\times 4$). The decoder has to process the requests in serial due to the variable size limitation: it extracts address, granularity and other messages of the first write request, after then it can calculate the length of write-data based on the granularity. Then it can retrieve the write data based on the length and advance to the next write request until all the write requests are retrieved.

3.3 Challenge of Designing a Message Interface Based Memory System

Besides valuable optimization benefit, message-based interface for memory system will introduce some challenges to all system levels that are concerned with memory. Many challenges remain to be solved. The following is an incomplete list.

1) *Complexity*. Message packet processing is more complex than processing a simple packet. Both the memory controller and the buffer scheduler need more complex logic to accomplish the task, e.g., longer queue management and complex consistence checking. Although logic is becoming cheaper, it still needs to investigate whether the cost, power consumption, and increased processing latency can be controlled within an acceptable level.

2) *ISA Extension*. To fully utilize the flexibility of messages, the processor needs to provide more semantic information along with read/write requests. This needs to bring some extensions to the ISA. For example, how to provide the size information for variable granularity memory requests; how to deliver process information such as thread ID, priority, timeout, and prefetch; how to generate active memory operation requests to the memory controller and the buffer scheduler.

3) *Cache Support*. To better support variable granularity memory accesses, the cache that supports variable-sized cache block is preferred though with difficulty. Sector cache for fine granularity and SPM (scratchpad memory) for large granularity can also be adopted with a redesign.

4) *Programming*. The semantic information may also be discovered and generated by user-level software and sent via message. Applications can be implemented with some hint API, or with the help of an aggressive compiler to generate MIMS special instructions automatically.

4 Experimental Setup

4.1 Simulator and Workloads

To evaluate the performance of MIMS, we have implemented a cycle-detailed MIMS simulator called MIMSim. We adopt DRAM modules (devices) based on DRAMSim2[44], which is a cycle accurate DDR2/3 memory system simulator. DRAMSim2 models all aspects of the memory controller and DRAM devices, including transaction queue, command queue, and read-return queue, address mapping scheme, DDR data/address/command bus contention, DRAM device power and timing, and row buffer management. The DRAM module is modified to support sub-rank and fine granularity memory access. We add re-order buffer (ROB) above the memory controller to make simulation more accurate. Channel interleaving address mapping is adopted as the default (baseline) configuration to maximum MLP (memory level parallelism). We adopt the FR-FCFS (First-Ready, First-Come, First-Served)[45] scheduling policy with closed-page row buffer management.

Pin[46] is used to collect memory access traces from various workloads running with 2∼16 threads. We choose several multi-thread memory intensive applications including BFS in Graph500[12], Canneal benchmark in PARSEC[47], Listrank[48], Pagerank benchmark in GraphLab[13], SSCA2[49], GUPS[14]. Table 2 lists the main characteristics of these workloads. We classify the workloads into two categories based on the memory access granularity: fine granularity (FINE: 0∼3), and middle granularity (MID: 3∼6). We do not investigate applications with coarse granularity since they can be treated efficiently already. Memory read and write requests are reported separately, including the read memory requests per kilo instruction (RPKI), the average read granularity (RG), the write memory requests per kilo instruction (WPKI), the average write granularity (WG), and the read/write ratio (RD/WT). The reason to separate the read and write characteristics is that we find the granularity distribution of read and write might be different for some FINE and MID workloads. Fig.5 shows the granularity distribution of these workloads. For example, in the Canneal workload, the rate of 1-granularity are about 72.85% for read requests, but it is about 97.59% for write requests. And

[12]http://www.graph500.org/, Dec. 2013.

[13]GraphLab: Distributed graph-parallel API. http://docs.graphlab.org/index.html, Dec. 2013.

[14]The RandomAccess benchmark: GUPS (giga updates per second). http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/, Dec. 2013.

in the Listrank workload, the rate of 2-granularity and 4-granularity is about 52.99% and 31.54% respectively for read requests, but they are about 76.51% and 0.90% respectively for write requests.

Table 2. Workloads Characteristics

Category	Benchmark	RPKI	RG	WPKI	WG	RD/WT
FINE	GUPS	69.67	1.78	69.62	1.78	1.00
FINE	SSCA2	20.89	1.68	20.42	1.56	1.02
FINE	Canneal	17.79	1.64	8.64	1.10	2.06
FINE	Pagerank	9.76	2.42	6.14	2.74	1.59
MID	Listrank	22.56	3.56	15.45	3.37	1.46
MID	BFS	22.36	3.10	2.44	3.49	9.16

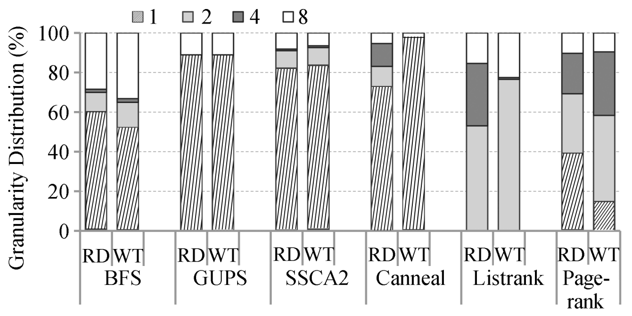


Fig.5. Read and write granularity distribution of FINE and MID memory-intensive workloads.

To collect exact granularity information for each off-chip memory request, we implement a 3-level cache simulator as a Pin-tool. The detailed configuration is listed in Table 3. We start the cache simulator after each workload enters into a representative region. After warm-up the cache simulator with 100 million memory requests, we collect memory traces with granularity and cache access type messages. For PARSEC benchmarks, we naturally choose the ROI (Region-of-Interest); for other benchmarks, we manually skip the initialization phase (such as graph-generation in BFS) and collect memory traces after meaningful work.

To make simulation more accurate with ROB, we separate instructions into two types: non-memory instructions and memory access instructions. And we collect the number of non-memory instructions between two adjacent memory access traces. Each non-memory instruction is considered to have a fixed latency when it is added in the ROB, and each memory access might have variable latency which depends on whether it is hit on any level of cache or it needs to access DRAM module.

Table 3. System Configurations

System Parameters	
Reorder buffer	2.7 GHz, 256-entry, maximum fetch/retire per cycle: 4/2, 5-pipeline
L1 cache	Private, 32 KB, 4-way, 64 B cache block, 4 CPU cycles hit
L2 cache	Private, 256 KB, 8-way, 64 B cache block, 10 CPU cycles hit
L3 cache	Shared, 16-way, 64 B cache line, 1 MB/core, 40 CPU cycles hit
Memory controller	2 buffer schedulers/MC, read/write queue: 64/64
Link bus	2.7 GHz, point-to-point, read/write bus width: 16/16
Buffer scheduler	FRFCFS[45], closed page, channel-interleave mapping
DRAM Parameters	
Memory	2 64-bit channels, 2 ranks/channel, 8 devices/rank, 8 sub-ranks/rank, x8-width sub-rank, 1 device/sub-rank
DRAM device	DDR3-1333 MHz, x8, 8 banks, 32 768 rows/bank, 1 024 columns/row, BL=8, 8 KB row buffer, time and power parameters from Micron 2 Gb SDRAM[15]

4.2 System Configurations

Table 3 lists the main parameter settings used in the cycle-detailed simulator. Note, the non-memory instruction latency and cache hit latencies listed here are used as the latency of an instruction needs to wait in the ROB (in MIMSim) before it can be committed. For example, an L2 cache hit memory access instruction can be committable only after 15 CPU cycles (10 + 5) when it is added in the ROB. The baseline memory system has two DDR3-1333MHZ channels with dual ranks in each channel. Each rank has 8 DRAM devices and each DRAM chip has 8 banks. We bypass the first 64 million memory traces for each core (thread), and simulate until all the threads have executed at least 100 million instructions.

To evaluate MIMS, we use the following memory system configuration:

- DDR: traditional DDRx (3) memory system with fixed coarse access granularity (cache line: 64 B), which is served as the baseline.
- BOB: buffer-on-board memory system, fixed coarse access granularity, 1 memory request (read/write) per packet, and simple packet format without any semantic information.
- MIMS_1 (MI_1): message interface based memory system, adopts sub-rank memory organization to support variable-granularity access, 1 request per packet, and contains granularity message in packet.
- MIMS_mul (MI_mul): message interface based memory system, supports variable-granularity access, and encapsulates multiple requests in a packet to reduce packet overhead.

[15]DDR3 SDRAM. http://download.micron.com/pdf/datasheets/dram/ddr3/2Gb_DDR3_SDRAM.pdf, Dec. 2013.

4.3 DRAM and Controller Power

We evaluate memory power consumption with DRAMsim2[44] power calculator, which uses the power model developed by Micron Corporation based on the transitions of each bank. The DRAM power is divided into 4 components: background, refresh, activation/precharge, and burst, where background and refresh power are often concluded as static power, activation/precharge and burst power are concluded as dynamic power. Besides DRAM devices, we also take the memory controller power into account, since it can contribute a significant amount to overall power consumption[13] (about 20%). In BOB and MIMS, the controller power is actually referred to simpler controller and buffer scheduler power respectively. For DDR, we adopt the controller power to 8.5 W; for BOB and MIMS, we adopt the intermediate controller power to 14 W as in [9]. The controller idle power is set to 50% of its peak power.

5 Experimental Results

In this section, we will present the evaluation of performance and power impacts of MIMS. We present simulation results of a 16-core system on FINE and MID granularity workloads. All the workloads are in multiple-thread mode, with each core running one thread. We use the total number of submitted instructions as the metric of performance.

Performance. Fig.6 shows the normalized performance speedup and effective bandwidth utilization of different memory systems, where the baseline is DDR. For these FINE or MID workloads, such as BFS, Canneal, GUPS, fine granularity memory access will benefit. The BOB performance degrades from 49.38% to 78.38%. This is because that BOB still uses coarse granularity access, and the intermediate controller would introduce packet overhead and extra processing latency. On the other side, MI_1 and MI_mul can improve the performance because they support variable-granularity. The normalized speedup of MI_1 ranges from 1.11 to 1.53, and it ranges from 1.29 to 2.08 for MI_mul. This indicates that integrating multiple requests in a packet can reduce packet overheard, thus improve memory performance. The effective bandwidth utilizations nearly have the same trend with the speedups in different memory systems. For DDR, they range from 15.58% to 31.55%; for BOB, the effective bandwidth utilization is decreased, since each memory request would introduce a packet overhead, along with the wasting bandwidth for transferring useless data in a cache block. MI_1 can eliminate wasting data but still suffer significant packet overhead for FINE granularity memory accesses. MI_mul can achieve the best efficiency bandwidth utilization, ranging from 21.15% to 44.49%.

Power. Fig.7 shows the memory power breakdown and the normalized EDP in different memory systems. The DRAM memory power can be divided into background, refresh, activation/precharge (Act/Pre), and read/write (Burst). The background power is due to the peripheral circuitry, and it is independent of memory access. The refresh power is due to DRAM cell leakage and refresh operations to maintain data integrity. The activation/precharge power is due to the memory accesses to different row buffers within DRAM chips. The read/write power is due to the column accesses to opened row buffers. Here we also consider the power of controller, which is due to performing memory schedule in the memory controller for DDR, and the added buffer scheduler for other memory systems (BOB, MIMS). The average total power for DDR is about 23.38 W, and BOB has a little more power (26.36 W), since the intermediate simple controller consumes more power than the on-chip memory controller. The DRAM power of them is nearly the same. MI_1 and MI_mul can effectively reduce the activation/precharge (Act/Pre) power because each (fine granularity) memory request only activates/precharges a sub-rank (one DRAM device in our work) with smaller row, and reduce the burst power because it only reads/writes the really useful part of data instead of a cache block (such

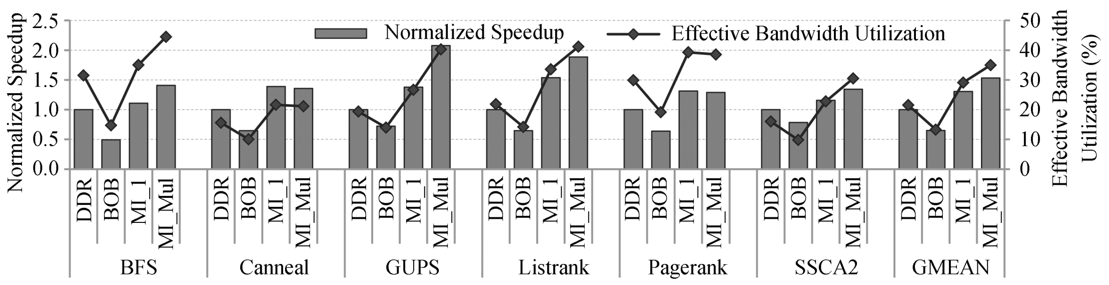


Fig.6. Normalized speedup and effective bandwidth utilization of different memory systems in 16-core configuration. The baseline is DDR. GMEAN: geometric mean.

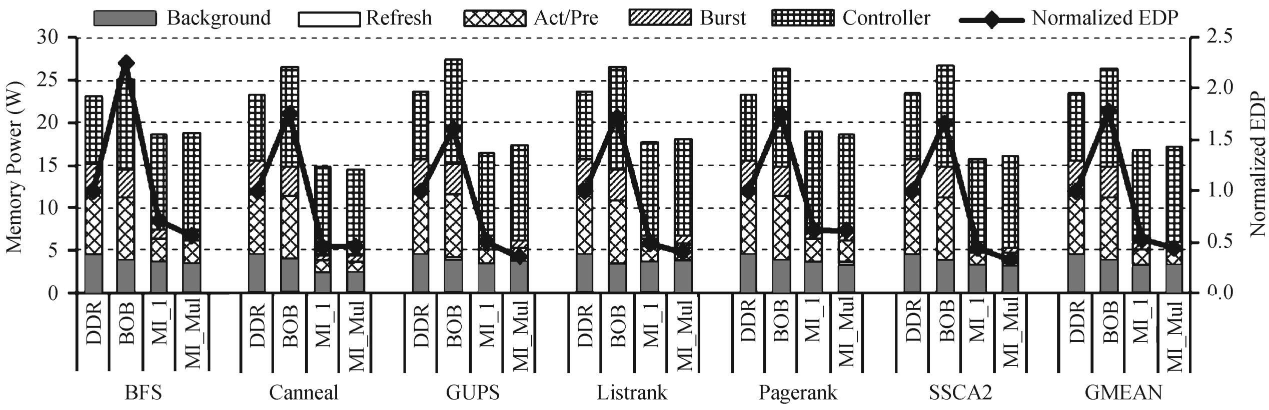


Fig.7. Memory power breakdown and the normalized EDP (the lower, the better) in 16-core configuration. GMEAN: geometric mean.

as 8 B data in 64 B cache block). Thus the power of MI_1 and MI_mul reduced to 16.90 W and 17.13 W respectively. The normalized EDP (Energy Delay Product) of BOB reduces about 1.78. This is mainly because the introduced packet-processing latency. MI_1 and MI_mul can improve EDP by about 0.53 and 0.44 respectively. This is because sub-rank can improve memory parallelism and thus efficiently reduce queuing delay.

Memory Latency. In DDR memory system, the memory latency is categorized into two major categories: queuing latency (or delay) and DRAM core access latency. The queuing latency represents the latency of a memory request waiting to be scheduled in the transaction queue (in memory controller), which has been proved to be the main component of memory latency in multi-core system[1]. The DRAM core access latency represents the latency of executing DDR commands of a memory request in DRAM devices. In MIMS, there is an additional scheduling latency, which represents the extra processing latency induced by the buffer scheduler. It includes the SerDes latency, scheduling latency, and packet encoding/decoding latency. The queuing latency occur both in memory controller (waiting to be packed) and in buffer scheduler (waiting to be issued to DRAM devices) in MIMS.

Fig.8 shows the memory latency breakdown in 16-core configuration. We can see that for these memory intensive workloads, the queuing latency dominates the overall memory latency, especially for GUPS and SSCA2 workload, which can achieve up to 1 185.67 ns and 933.0 ns respectively in DDR memory system, meanwhile the DRAM core access latency is only 22.22 ns. The reason for it is that these two workloads suffer high MPKI as shown in Table 2 and the traditional DDR memory system fails to serve them due to its limited memory level parallelism. However, the queuing latency can be reduced significantly in MIMS. For instance, it reduces to 234.81 ns for GUPS and 147.41 ns for SSCA2. This is because MIMS adopts sub-rank and it can provide more MLP since each narrow sub-rank can be accessed in parallel. Even though the intermediate buffer scheduler will induce extra scheduling latency, the overall memory latency is reduced for all workloads.

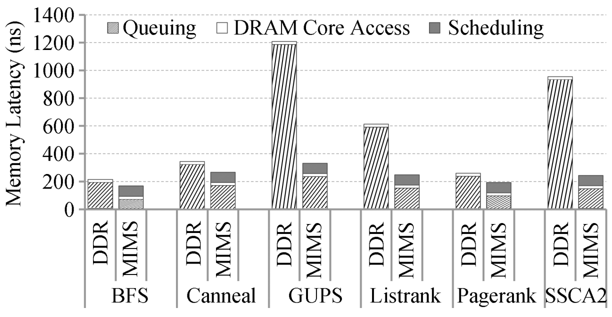


Fig.8. Memory latency breakdown in 16-core configuration.

Latency Effect. The buffer scheduler will introduce extra packet processing latency to memory requests, since the memory controller needs firstly to send requests to the buffer scheduler in packets. The extra latency includes: packing multiple requests in the memory controller, SerDes transition, packet transferring over serial link bus, and packet decoding. Due to different implementations and technics, the introduced latency of buffer scheduler might have a range of possibilities. In this subsection, we vary the latency from 0 (perfect) to 200 CPU cycles with a step of 40 CPU cycles to study how the introduced latency would affect the overall memory system performance. Fig.9 shows the results.

We can see that the latency of the buffer scheduler has a significant impact on the MIMS performance, and the impact is different for different workloads. SSCA2 has the largest slowdown at a rate about 1.14 on processing latency of every 40 more CPU cycles, and the normalized speedup (based on DDR) reduces from 2.12 (with no extra cycle) to 1.09 (with 200 more cycles).

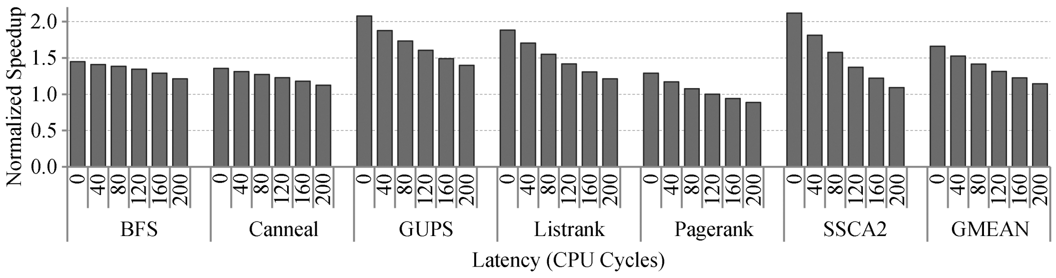


Fig.9. Normalized speedup of MIMS as the introduced packet-processing latency varies from 0 to 200 CPU cycles. GMEAN: geometric mean.

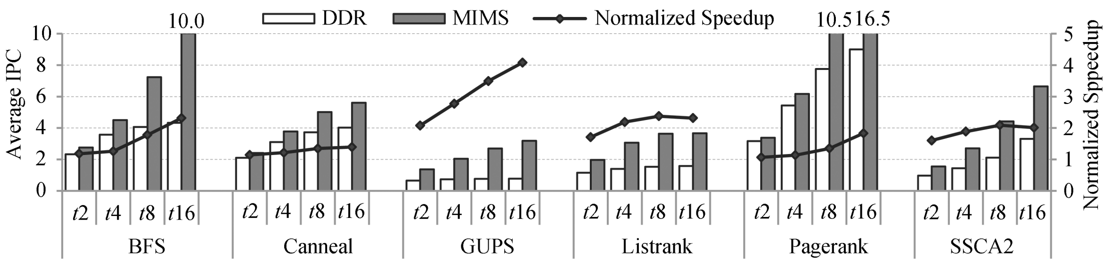


Fig.10. Normalized speedup varied as the number of threads increasing from 2 to 16.

For Pagerank, the performance even reduces to 0.89, which is worse than the baseline DDR. All workloads still have good speedup with 100-cycle delay, which means that with subtle design for the buffer scheduler to reduce its processing latency, MIMS can improve the overall system performance.

Fig.10 evaluates this problem with another view. As we varying the number of threads from 2 to 16, the speedup of MIMS over DDR3 will increase significantly. It means that the more parallelism in the workloads, the better acceleration will be achieved. This implies that providing more parallelism might hide the extra latency effectively.

In MIMS, we adopt the adaptive packet format, which can provide more opportunities for further optimizations. In the following subsections, we will introduce two optimizations on packets as shown in our previous work[50]: address compression for multiple memory requests in a packet, and merging multiple continuous small-size memory requests into a large-size memory request (which can further reduce packet overhead).

Address Compression. Table 4 shows the result of address compression for multiple memory requests in a packet. We use a base address table to save a series of base addresses and transmit the new updated base address in packet to synchronize the base address table. Although the base address table needs some extra storage, this strategy brings the best effect of reducing the bits for transmission. The compress ratio column shows such effect. The ADDR fields in a packet are shorten from 44.2% to 69.4% as before. Shorter ADDR length is reflected in the bus utilization reduction. From this table, we also find that address compression has better results on workloads that have mainly fine-grained memory access, which infers the proportion of addresses is large in packet.

Table 4. Compress Ratio and Memory Bus Utilization Reduction of Address Compression in MIMS

Workload	Compress Ratio	Bus Utilization Reduction (%)
Streamcluster	1.61	7.93
STREAM	2.26	3.71
PerM	2.01	4.36
ScaleParC	2.26	12.25
SPECCPU/437	1.44	4.93
SPECCPU/458	2.14	7.46

Continuous Memory Requests Combining. We adopt good memory locality with coarse granularity benchmarks to evaluate the performance improvement, including STREAM benchmark, BT, FT, UA, SP, and MG benchmarks from the NAS Parallel Benchmarks (NPB), the perM benchmark, and the Streamcluster benchmark from PARSEC. Fig.11 shows the result of memory access granularity and performance improvement after combining multiple small-size continuous requests into a large-size request. We can find that some workloads have really good memory access locality (or continuity) so that a large number of large-granularity memory requests are successfully generated after combining the continuous memory requests. The contention on the memory bus is reduced by this combination and the row buffer utilization is improved. Thus the overall performance of the memory system can be improved. In Fig.11, we can see that the average performance improvement is about 17.0%, while the maximal improvement is about 24.9% for FT workload.

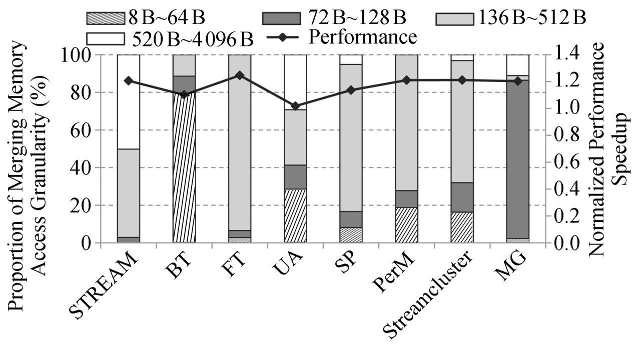


Fig.11. Memory access granularity and performance improvement after combining continuous memory requests.

6 Conclusions and Future Work

In this paper, a message interface based memory system (MIMS) was proposed. By decoupling memory access with memory organization and associating semantic information with memory request, MIMS provides new opportunities to solve many existing memory problems. Experimental results show that MIMS is able to improve parallelism and bandwidth utilization for fine granularity applications and reduce power consumption at the same time. It was also showed that although MIMS would introduce extra access latency for individual operation, the increased parallelism could reduce the queuing latency and result in a overall latency improvement.

Using message interface instead of traditional bus might open a new road for memory system innovations. In the future we will extend MIMS to support more operations and investigate on various implementation issues.

References

[1] Udipi A N, Muralimanohar N, Chatterjee N *et al.* Rethinking DRAM design and organization for energy-constrained multi-cores. In *Proc. the 37th Annual Int. Symposium on Computer Architecture*, Jun. 2010, pp.175-186.

[2] Wulf W A, McKee S A. Hitting the memory wall: Implications of the obvious. *SIGARCH Computer Architecture News*, 1995, 23(1): 20-24.

[3] Rogers B M, Krishna A, Bell G B *et al.* Scaling the bandwidth wall: Challenges in and avenues for CMP scaling. In *Proc. the 36th Annual Int. Symposium on Computer Architecture*, Jun. 2009, pp.371-382.

[4] Yoon D H, Jeong M K, Erez M. Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput. In *Proc. the 38th Annual Int. Symposium on Computer Architecture*, Jun. 2011, pp.295-306.

[5] Ferdman M, Adileh A, Kocberber O *et al.* Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *Proc. the 17th Int. Conf. Architectural Support for Programming Languages and Operating Systems*, Mar. 2012, pp.37-48.

[6] Lotfi-Kamran P, Grot B, Ferdman M *et al.* Scale-out processors. In *Proc. the 39th Int. Symposium on Computer Architecture*, Jun. 2012, pp.500-511.

[7] Yoon D H, Jeong M K, Sullivan M *et al.* The dynamic granularity memory system. In *Proc. the 39th Int. Symposium on Computer Architecture*, Jun. 2012, pp.548-559.

[8] Fredriksson H, Svensson C. Improvement potential and equalization example for multidrop DRAM memory buses. *IEEE Transactions on Advanced Packaging*, 2009, 32(3): 675-682.

[9] Cooper-Balis E, Rosenfeld P, Jacob B. Buffer-on-board memory systems. In *Proc. the 39th Int. Symposium on Computer Architecture*, Jun. 2012, pp.392-403.

[10] Lee B C, Ipek E, Mutlu O *et al.* Architecting phase change memory as a scalable dram alternative. In *Proc. the 36th Annual Int. Symposium on Computer Architecture*, Jun. 2009, pp.2-13.

[11] Udipi A N, Muralimanohar N, Balsubramonian R *et al.* Combining memory and a controller with photonics through 3D-stacking to enable scalable and energy-efficient systems. In *Proc. the 38th Annual Int. Symposium on Computer Architecture*, Jun. 2011, pp.425-436.

[12] Barroso L A, Höelzle U. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines (1st edition). Morgan and Claypool Publishers, 2009.

[13] Deng Q Y, Meisner D, Ramos L *et al.* Memscale: Active low-power modes for main memory. In *Proc. the 16th Int. Conf. Architectural Support for Programming Languages and Operating Systems*, Mar. 2011, pp.225-238.

[14] Li S, Chen K, Hsieh M Y *et al.* System implications of memory reliability in exascale computing. In *Proc. the 2011 Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, Nov. 2011, Article No.46.

[15] Yoon D H, Muralimanohar N, Chang J C *et al.* FREE-p: Protecting non-volatile memory against both hard and soft errors. In *Proc. the 17th IEEE Int. Symposium on High Performance Computer Architecture*, Feb. 2011, pp.466-477.

[16] Draper J, Chame J, Hall M *et al.* The architecture of the DIVA processing-in-memory chip. In *Proc. the 16th Int. Conf. Supercomputing*, Jun. 2002, pp.14-25.

[17] Fang Z, Zhang L X, Carter J B *et al.* Active memory operations. In *Proc. the 21st Annual Int. Conf. Supercomputing*, Jun. 2007, pp.232-241.

[18] Fang Z, Zhang L X, Carter J B *et al.* Active memory controller. *J. Supercomput.*, 2012, 62(1): 510-549.

[19] Lynch B, Kumar S. Smart memory. In *Hot Chips: A Symposium on High Performance Chips*, Aug. 2010. http://www.hotchips.org/wp-content/uploads/hc_archives/hc22/HC22.23.325-1-Kumar-Smart-Memory.pdf, Feb. 2014.

[20] Ware F A, Hampel C. Improving power and data efficiency with threaded memory modules. In *Proc. the 24th Int. Conf. Computer Design*, Oct. 2006, pp.417-424.

[21] Ahn J H, Leverich J, Schreiber R S *et al.* Multicore DIMM: An energy efficient memory module with independently controlled DRAMs. *Computer Architecture Letters*, 2009, 8(1): 5-8.

[22] Ahn J H, Jouppi N P, Kozyrakis C *et al.* Future scaling of processor-memory interfaces. In *Proc. the Conf. High Performance Computing Networking, Storage and Analysis*, Nov. 2009, Article No.42.

[23] Zheng H Z, Lin J, Zhang Z *et al.* Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In *Proc. the 41st Annual IEEE/ACM Int. Symposium on Microarchitecture*, Nov. 2008, pp.210-221.

[24] Fang K, Zheng H Z, Zhu Z C. Heterogeneous mini-rank: Adaptive, power-efficient memory architecture. In *Proc. the 39th Int. Conf. Parallel Processing*, Sept. 2010, pp.21-29.

[25] Zhang G F, Wang H D, Chen X K *et al.* Heterogeneous multi-channel: Fine-grained DRAM control for both system performance and power efficiency. In *Proc. the 49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, Jun. 2012, pp.876-881.

[26] Cooper-Balis E, Jacob B. Fine-grained activation for power reduction in DRAM. *IEEE Micro*, 2010, 30(3): 34-47.

[27] Brewer T M. Instruction set innovations for the convey HC-1 computer. *IEEE Micro*, 2010, 30(2): 70-79.

[28] Abts D, Bataineh A, Scott S *et al.* The cray blackwidow: A highly scalable vector multiprocessor. In *Proc. the 2007 ACM/IEEE Conf. Supercomputing*, Nov. 2007, Article No.17.

[29] Chen L, Cao Y N, Zhang Z. E3CC: A memory error protection scheme with novel address mapping for subranked and low-power memories. *ACM Transactions on Architecture and Code Optimization*, 2013, 10(4): Article No.32.

[30] Yoon D H, Erez M. Virtualized and flexible ECC for main memory. In *Proc. the 15th Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, Mar. 2010, pp.397-408.

[31] Udipi A N, Muralimanohar N, Balsubramonian R *et al.* LOT-ECC: Localized and tiered reliability mechanisms for commodity memory systems. In *Proc. the 39th Annual Int. Symposium on Computer Architecture*, Jun. 2012, pp.285-296.

[32] Zheng H Z, Lin J, Zhang Z *et al.* Decoupled DIMM: Building high-bandwidth memory system using low-speed DRAM devices. In *Proc. the 36th Annual Int. Symposium on Computer Architecture*, Jun. 2009, pp.255-266.

[33] Yoon D H, Chang J C, Muralimanohar N *et al.* BOOM: Enabling mobile memory based low-power server DIMMs. In *Proc. the 39th Int. Symposium on Computer Architecture*, Jun. 2012, pp.25-36.

[34] Kalla R, Sinharoy B, Starke W J *et al.* Power7: IBM's next-generation server processor. *IEEE Micro*, 2010, 30(2): 7-15.

[35] Van Huben G A, Lamb K D, Tremaine R B *et al.* Server-class DDR3 SDRAM memory buffer chip. *IBM Journal of Research and Development*, 2012, 56(1.2): Article No.3.

[36] Fang K, Chen L, Zhang Z *et al.* Memory architecture for integrating emerging memory technologies. In *Proc. the 2011 Int. Conf. Parallel Architectures and Compilation Techniques*, Oct. 2011, pp.403-412.

[37] Ham T J, Chelepalli B K, Xue N *et al.* Disintegrated control for energy-efficient and heterogeneous memory systems. In *Proc. the 19th Int. Symposium on High Performance Computer Architecture*, Feb. 2013, pp.424-435.

[38] Hall M, Kogge P, Koller J *et al.* Mapping irregular applications to DIVA, a PIM-based data-intensive architecture. In *Proc. the 1999 ACM/IEEE Conf. Supercomputing (CDROM)*, Jan. 1999, Article No.57.

[39] Qureshi M K, Srinivasan V, Rivers J A. Scalable high performance main memory system using phase-change memory technology. In *Proc. the 36th Annual Int. Symposium on Computer Architecture*, Jun. 2009, pp.24-33.

[40] Zhou P, Zhao B, Yang J *et al.* A durable and energy efficient main memory using phase change memory technology. In *Proc. the 36th Annual Int. Symposium on Computer Architecture*, Jun. 2009, pp.14-23.

[41] Zhang W Y, Li T. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures. In *Proc. the 18th Int. Conf. Parallel Architectures and Compilation Techniques*, Sept. 2009, pp.101-112.

[42] Stuecheli J, Kaseridis D, Daly D *et al.* The virtual write queue: Coordinating DRAM and last-level cache policies. In *Proc. the 37th Annual Int. Symposium on Computer Architecture*, Jun. 2010, pp.72-82.

[43] Chatterjee N, Muralimanohar N, Balasubramonian R *et al.* Staged reads: Mitigating the impact of DRAM writes on DRAM reads. In *Proc. the 18th Int. Symposium on High-Performance Computer Architecture*, Feb. 2012, pp.41-52.

[44] Rosenfeld P, Cooper-Balis E, Jacob B. DRAMSIM2: A cycle accurate memory system simulator. *Computer Architecture Letters*, 2011, 10(1): 16-19.

[45] Rixner S, Dally W J, Kapasi U J *et al.* Memory access scheduling. In *Proc. the 27th Annual Int. Symposium on Computer Architecture*, Jun. 2000, pp.128-138.

[46] Luk C K, Cohn R, Muth R *et al.* Pin: Building customized program analysis tools with dynamic instrumentation. In *Proc. the 2005 ACM SIGPLAN Conf. Programming Language Design and Implementation*, Jun. 2005, pp.190-200.

[47] Bienia C, Kumar S, Singh J P *et al.* The PARSEC benchmark suite: Characterization and architectural implications. In *Proc. the 17th Int. Conf. Parallel Architectures and Compilation Techniques*, Oct. 2008, pp.72-81.

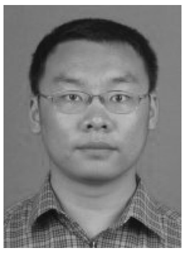
[48] Bader D A, Cong G J, Feo J. On the architectural requirements for efficient execution of graph algorithms. In *Proc. the 2005 Int. Conf. Parallel Processing*, Jun. 2005, pp.547-556.

[49] Bader D A, Madduri K. Design and implementation of the HPCS graph analysis benchmark on symmetric multiprocessors. In *Proc. the 12th Int. Conf. High Performance Computing*, Dec. 2005, pp.465-476.

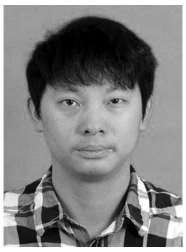
[50] Lu T Y, Chen L C, Chen M Y. Achieving efficient packet-based memory system by exploiting correlation of memory requests. In *proc. Design, Automation & Test in Europe*, Mar. 2014, to be appeared.

Li-Cheng Chen is a Ph.D. candidate of Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing. Previously, he obtained the B.S. degree from University of Science and Technology of China (USTC) in computer science, Hefei, in 2008. His current research focuses on computer architecture, memory architecture and optimization, system performance evaluation and optimization, and memory management.

Ming-Yu Chen received his B.S. degree from University of Science and Technology of China in 1994 and M.S. and Ph.D. degrees both in computer science from the CAS in 1997 and 2000, respectively. He is a processor in ICT, CAS. His research interests include computer architecture, operating system, and algorithm optimization for high performance computers. He is a member of CCF, ACM, IEEE.



Yuan Ruan received his B.S. degree from Shandong University in 2004 and Ph.D. degree in computer science from the CAS in 2010. He is an engineer in ICT, CAS. His current research interests include computer architecture, memory system and hardware acceleration architecture. He is a member of CCF, ACM.



Yong-Bing Huang is a Ph.D. candidate of ICT, CAS, Beijing. Previously, he obtained the B.S. degree from Wuhan University in computer science, in 2008. His current research interests focus on computer architecture, operating system, system performance modeling and evaluation, and hardware or architecture support for applications.

Ze-Han Cui is a Ph.D. candidate of ICT, CAS, Beijing. Previously, he obtained the B.S. degree from USTC in 2009. His current research interests include computer architecture, memory architecture and power, system performance modeling and evaluation.

Tian-Yue Lu is a M.S. student of ICT, CAS, Beijing. Previously, he obtained the B.S. degree from USTC in 2011. His current research interests include computer architecture, memory architecture and optimization, system performance modeling and evaluation.

Yun-Gang Bao received his B.S. degree from Nanjing University in 2003 and Ph.D. degree in computer science from the CAS in 2008. He is an associate professor in ICT, CAS. From 2010 to 2012, he was a postdoctoral researcher in Department of Computer Science, Princeton University. His current research interests include computer architecture, operating system, system performance modeling and evaluation. He is a member of CCF, ACM, IEEE.