Architecture Support for Emerging Memory Technologies

BY

KUN FANG
B.E., Huazhong University of Science and Technology, 2005
M.E., Huazhong University of Science and Technology, 2007

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2013

Chicago, Illinois

Defense Committee:
Zhichun Zhu, Chair and Advisor
Shantanu Dutt
Ashfaq Khokhar
Wenjing Rao
Ajay Kshemkalyani, Computer Science

To my parents,

Hongsheng Fang and Xiaoping Li,

my wife,

Runzhe Tao,

whose love encourages me towards excellence.

ACKNOWLEDGMENTS

I would like to thank Professor Zhichun Zhu, for her guidance, her encouragement, and her support during my Ph.D. research. I am also indebted to Dr. Shantanu Dutt and Dr. Ashfaq Khokhar for teaching the courses of electrical and computer engineering that helped me understand the circuits better. Also, I would like to thank Dr. Wenjing Rao for the group presentations and discussions and Dr. Ajay Kshemkalyani for the insightful comments on my research. In particular, I would like to thank Dr. Zhao Zhang and Long Cheng of Iowa State University. Our collaboration builds the foundation of my several research projects. I would also like to thank Dr. Hongzhong Zheng who helped me to quickly pass the starting phase of Ph.D..

A number individuals were also extremely helpful when I have to live in a totally new country for study. Han Liang showed me around Chicago. Hua Wei, Bo Ling and Shen Min were my roommates. Yehua Su, Yu Liu, Chi Zhang, Jinghu Li, Jingye Xu, Jiang Lin, Suyu Zhang, Kun Ma, Soumya Banerjee and Hui Lin are the names flashed across my mind when I pick the word “friendship”.

KF

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF ABBREVIATIONS

CPU	Central Processing Unit
DRAM	Dynamic Random Access Memory
SDRAM	Synchronous Dynamic Random Access Memory
DDR	Double Data Rate
GDDR	Graphics Double Data Rate
XDR	eXtreme Data Rate
DIMM	Dual In-line Memory Module
FB-DIMM	Fully Buffered Dual In-line Memory Module
SPEC	Standard Performance Evaluation Corporation
PCM	Phase Change Memory
MLC	Multi-Level Cell
MRAM	Magnetic Random Access Memory
STT-MRAM	Spin-Torque Transfer Magnetic Random Access Memory
EDP	Energy Delay Product
CRA	Conservertive Row Activation
UniMA	Universal Memory Architecture

LIST OF ABBREVIATIONS (Continued)

RAS	Row Address Strobe
CAS	Column Address Strobe
AMB	Advanced Memory Buffer
MRB	Mini-Rank Buffer
BRT	Bus Reservation Table
ACT	Activation Access
PRE	Precharge Access
COL	Column Access
CKE	Clock Enable
PDE	Power-Down Enter
PDX	Power-Down eXit
SRE	Self-Refresh Enter
SRX	Self-Refresh eXit
REF	Refresh
DLL	Delay Locked Loop
MEM	Memory Intensive Workload
ILP	CPU Intensive Workload
MIX	Mixed Workload

LIST OF ABBREVIATIONS (Continued)

IPC	Instruction Per Cycle
FR-FCFS	First Ready - First Come First Serve
TMT	Type Mapping Table
MEMTMT	Type Mapping Table in Memory Controller
OSTMT	Type Mapping Table in Operating System
UDIC	Unified Dual In-line Memory Module Interface Chip

SUMMARY

Memory system is responsible to store all the information used and produced by the CPU in the Von Neumann architecture. As the technology advances for the processors, the growing disparity of speed between CPU and main memory becomes so large that the so called "memory wall" becomes the major bottleneck of a modern computer system. In recent decades, the CPU speed keeps improving at an annual rate of 60% but the memory speed only improves at 10%. Although after the year 2000, CPU speed improvements slowed down due to thermal constraints, the chip geometries shrink steadily. We now are entering a new era that the number of hardware threads per socket increases dramatically (33). With the wide spread use of multi-core and many-core processors, current memory system designers try to push the memory bandwidth and capacity to meet the user demands. A negative side-effect is the continuous increase on memory power consumption. Listed in a publication of Intel (34), the power consumption of processor can vary from 45W to 200W and on average, DDR3 power can be from 2W to 13W for a small configuration (2GB to 4GB). As the memory capacity increases to 32 GB or 64GB and the memory speed is forecasted to be at 1866MHz and 2133MHz, the power consumption of memory system will surpass the CPU power.

In addition, main memory system design is severely limited by the rigid architecture that requires the memory controller to track the internal status of all memory devices (chips) and schedule the timing of all device operations. As a result, DRAM memory system is heading to the scalability wall. New memory technologies, for instance, Phase-Change Memory

(PCM), Spin Torque Transfer Magnetic RAM (STT-MRAM), are promising alternatives to the DRAM technology in future memory systems. Although those technologies have better energy-efficiency and scalability than DRAM, they also suffer from low write-endurance and long write-latency. Thus, new memory architectures are needed for supporting future memory systems and balancing among performance, energy-efficiency, capacity and lifetime.

In this thesis, we address the memory power and scalability issues for multi-core and many-core systems, and give answers on how to reduce memory power consumption, how to get the best tradeoff between performance and memory power and eventually how to combine the emerging and current memory technologies to improve memory system efficiency. The works done and presented in this thesis form a systematic support for improving memory system efficiency at the architecture level by three steps. Firstly, a new DRAM scheduling algorithm called Conservative Row Activation is proposed to make the DRAM more energy-efficient by allowing memory ranks stay at a low-power mode longer if the data bus ownership cannot be acquired immediately after row activation finishes (25). Secondly, we present a heterogeneous mini-rank memory architecture that allows concurrently running applications to have different sub-rank widths based on their memory access behavior (24). By dynamically assigning and changing the sub-rank configurations, the balance can be achieved between the performance and power saving, and large performance loss can be avoided (23). Lastly, we build a new memory architecture framework called Universal Memory Architecture (UniMA) (22) that can support different memory technologies in a computer system by decoupling the scheduling of

device operations from memory controller. A bridge chip is added to each memory module to perform device-specific scheduling locally.

Our first work, the Conservative Row Activation is based on the observation that due to the increasing importance of memory-level parallelism, most memory scheduling schemes eagerly exploits such parallelism to optimize performance. A common policy used by memory controllers today is, whenever possible, always trying to open memory banks for pending requests to maximize bank-level parallelism and throughput. However, we find out that this is neither power optimal nor necessary for maintaining performance because usually many banks are open while waiting for the data bus ownership. The purpose of "Conservative Row Activation" memory access scheduling scheme is to improve memory system power efficiency with minimal performance impact. Instead of activating a bank at the earliest possible time, it activates a bank only when the follow-up column access will not wait for the data bus ownership. The scheme monitors the data bus transactions and reserves bus slots for each pending column command at the earliest possible time. Then the optimal time to issue the corresponding row activation is calculated as tRCD cycles before the column command. In this way, more banks can stay in idle state longer and ranks will get a better chance to stay in the low-power, precharge state to save power. In addition, because the column commands can still be issued almost at the same time as before, the performance impact would be minimal.

Our second work improves a recent proposed mini-rank memory architecture. The mini-rank (96) memory system achieves significant memory power reduction with a slight increase of the data transfer time. It uses a bridge chip called MRB (Mini-Rank Buffer) on each DIMM,

between the DRAM devices and the DDRx bus, to break conventional x64 ranks into x32, x16 or even smaller ranks to save memory power. Although the original mini-rank design works well for most workloads, we further observe that it has a limitation: it applies a fixed and unified partition from ranks to mini-ranks on all applications. Some applications will suffer a great performance loss under narrow mini-rank configurations, such as x8 mini-ranks; while others may lose power saving opportunities under wide mini-rank configurations, such as x32 mini-ranks. To address this issue, we propose a heterogeneous mini-rank scheme (23) that allows each application to have its own mini-rank configuration within a single memory system in order to approach the optimal power-performance trade-off. A latency-sensitive application will be assigned to a wide mini-rank configuration for maintaining its performance; while a latency-insensitive application will be assigned to a narrow mini-rank configuration for memory power saving. We find out that the memory bandwidth requirement of an application can be used as a guidance to select its near-optimal configuration. It is simple and only introduces small overhead in both software and hardware implementations.

Our third work is to provide architecture support for new memory technologies. Recent research shows a trend of using new memory technologies to build the main memory system. But without universal interoperability, future systems may have to use different memory controllers for various types of memory modules, which is infeasible in economic sense. Furthermore, future processors are likely to use integrated memory controllers, which means there would be many subtypes of processors. We propose Universal Memory Architecture (UniMA) to enable universal interoperability between all major processors and memory modules made by emerging

memory technologies as well as DRAM. The UniMA is a framework of memory architecture rather than a particular implementation. It is an advanced memory organization with a new memory access protocol that offloads the scheduling of device-level operations to each memory module. Each memory module embeds a bridge chip that performs local management, e.g., for DRAM devices, to schedule device operations including precharge, activation and read/write. The memory controller may still perform memory access reordering, i.e. reorder memory requests based on some scheduling schemes, but without timing constraints. The bridge chip is an extra chip on the memory module like those on the FB-DIMM (86), Mini-Rank (96) and Decoupled DIMM (97), or the bottom chip in future stacked 3-D memory modules. Since the controller no longer has the full knowledge of each device's status, a new protocol is proposed for the communication between the controller and modules to avoid bus contentions. The controller sends generic commands such as reads or writes instead of device-specific ones to modules; a module with ready data will raise its readiness via some additional signal lines; and a token-based approach is used to grant bus ownership to one ready module and avoid contentions.

Throughout this thesis, we demonstrate that our schemes can save DRAM power, provide optimal energy efficiency for mini-rank kind of design and integrate diverse memory technologies into one memory system with small overhead. A combination of the techniques in this thesis is straightforward and would be able to show a path to utilize both dominant and future memory technologies in a single computer system.

CHAPTER 1

INTRODUCTION

Memory system is responsible to store all the information used and produced by the CPU in the Von Neumann architecture. As the technology advances for the processors, the growing disparity of speed between CPU and main memory becomes so large that the so-called "memory wall" becomes the major bottleneck of a modern computer system. In recent decades, the CPU speed keeps improving at an annual rate of 60% but the memory speed only improves at 10%. Although after the year 2000, CPU speed improvements slowed down due to thermal constraints, the chip geometries shrink steadily. We now are entering a new era that the number of hardware threads per socket increases dramatically (33). With the wide spread use of multi-core and many-core processors, current memory system designers try to push the memory bandwidth and capacity to meet the user demands. A negative side-effect is the continuous increase on memory power consumption. Listed in a publication of Intel (34), the power consumption of processor can vary from 45W to 200W and on average, DDR3 power can be from 2W to 13W for a small configuration (2GB to 4GB). As the memory capacity increases to 32 GB or 64GB and the memory speed is forecasted to be at 1866MHz and 2133MHz, the power consumption of memory system will surpass the CPU power.

In addition, main memory system design is severely limited by the rigid architecture that requires the memory controller to track the internal status of all memory devices (chips) and schedule the timing of all device operations. As a result, DRAM memory system is head-

ing to the scalability wall. New memory technologies, for instance, Phase-Change Memory (PCM), Spin Torque Transfer Magnetic RAM (STT-MRAM), are promising alternatives to the DRAM technology in future memory systems. Although those technologies have better energy-efficiency and scalability than DRAM, they also suffer from low write-endurance and long write-latency. Thus, new memory architectures are needed for supporting future memory systems and balancing among performance, energy-efficiency, capacity and lifetime.

The goal of this thesis is to propose a systematic way to optimize the memory system and balance among the three most important characters: power, performance and capacity. The recently developed memory technologies also provide interesting opportunity and material to improve the memory system. Our research presented in this Ph.D. thesis addresses the goal through a series of implementations. Firstly, a new DRAM scheduling algorithm called Conservative Row Activation is proposed to make the DRAM more energy-efficient by allowing memory ranks stay at a low-power mode longer if the data bus ownership cannot be acquired immediately after row activation finishes (25). Secondly, we present a heterogeneous mini-rank memory architecture that allows concurrently running applications to have different sub-rank widths based on their memory access behavior (24). By dynamically assigning and changing the sub-rank configurations, the balance can be achieved between the performance and power saving, and large performance loss can be avoided (23). Lastly, we build a new memory architecture framework called Universal Memory Architecture (UniMA) (22) that can support different memory technologies in a computer system by decoupling the scheduling of device

operations from memory controller. A bridge chip is added to each memory module to perform device-specific scheduling locally.

Chapter 3 starts the whole research by reducing the conventional DDR3 memory system power consumption. A Conservative Row Activation algorithm is proposed. It is based on the observation that due to the increasing importance of memory-level parallelism, most memory scheduling schemes eagerly exploits such parallelism to optimize performance. A common policy used by memory controllers today is, whenever possible, always trying to open memory banks for pending requests to maximize bank-level parallelism and throughput. However, we find out that this is neither power optimal nor necessary for maintaining performance because usually many banks are open while waiting for the data bus ownership.

The purpose of "Conservative Row Activation" memory access scheduling scheme is to improve memory system power efficiency with minimal performance impact. Instead of activating a bank at the earliest possible time, it activates a bank only when the follow-up column access will not wait for the data bus ownership. The scheme monitors the data bus transactions and reserves bus slots for each pending column command at the earliest possible time. Then the optimal time to issue the corresponding row activation is calculated as tRCD cycles before the column command. In this way, more banks can stay in idle state longer and ranks will get a better chance to stay in the low-power, precharge state to save power. In addition, because the column commands can still be issued almost at the same time as before, the performance impact would be minimal.

We build a detailed memory model and integrate it to the MARSS (63) simulator. Running the quad-core workloads with SPEC2006 (81) benchmark, the experiment indicates that our proposed scheme can effectively improve memory system power efficiency with little performance impact. The memory power is saved by 5.6%, 3.3% and 5.8% on average for MEM, ILP and MIX workloads, respectively (up to 7.0%). The performance is even slightly improved for MEM workloads because our scheme can slightly improve the bus utilization by giving column access commands higher priority than others. The weighted speedup for MEM workloads is improved by 0.3% on average (up to 1.0%). For ILP and MIX workloads, the performance loss is less than 1.0%. The power saving is because our algorithm can put memory ranks in low power modes longer. So the background power contributes the most of the power reduction. This result also shows that our scheduling scheme can retain the performance effectively. This is because although the activation is delayed, the scheme looks into the future to place column commands wisely and the latency of a request is determined by when its column access is performed not when its row activation is performed. This mechanism does not change the baseline scheduling sequence of data transfer, hence the performance is kept the same.

After the successful attempt to save memory power, we further extend the original mini-rank design to make it dynamically reconfigurable. This extension puts us one step further towards our research goal. Now we can balance between performance and power to get the best energy-delay product. In Chapter 4, we discuss the heterogeneous mini-rank design based on a recent mini-rank (96) memory system for performance/power tradeoff, which achieves significant memory power reduction with a slight increase of the data transfer time. It uses

a bridge chip called MRB (Mini-Rank Buffer) on each DIMM, between the DRAM devices and the DDRx bus, to break conventional x64 ranks into x32, x16 or even smaller ranks to save memory power. Although the original mini-rank design works well for most workloads, we further observe that it has a limitation: it applies a fixed and unified partition from ranks to mini-ranks on all applications. Some applications will suffer a great performance loss under narrow mini-rank configurations, such as x8 mini-ranks; while others may lose power saving opportunities under wide mini-rank configurations, such as x32 mini-ranks. To address this issue, we propose a heterogeneous mini-rank scheme (23) that allows each application to have its own mini-rank configuration within a single memory system in order to approach the optimal power-performance trade-off. A latency-sensitive application will be assigned to a wide mini-rank configuration for maintaining its performance; while a latency-insensitive application will be assigned to a narrow mini-rank configuration for memory power saving. We find out that the memory bandwidth requirement of an application can be used as a guidance to select its near-optimal configuration. Then we further extend our heterogeneous mini-rank scheme by making the bandwidth detection for each application online to predict and dynamically assign and reassign of mini-rank configuration. The heterogeneous mini-rank scheme can provide near optimal performance/power trade-off, and avoid big performance loss for workloads with diverse memory access behavior. The heterogeneous mini-rank configuration selection is based on applications run-time memory bandwidth usage. It is simple and only introduces small overhead in both software and hardware implementations.

For heterogeneous workloads containing applications with diverse memory access behaviors, on average, the heterogeneous mini-rank can reduce the memory power by 18.6% (up to 38.0%) with the performance loss of 2.4% (up to 7.4%), compared with a conventional memory system. In comparison, the x32 homogeneous mini-rank can only save memory power by 12.6% on average (up to 25.4%); while the x8 homogeneous mini-rank will cause the performance loss by 8.1% on average (up to 19.3%). Compared with the x16 homogeneous mini-rank configuration, the heterogeneous mini-rank can also reduce the EDP (energy-delay product) by up to 9.6%.

In Chapter 5, we unify the dominant DDR technology and emerging memory technologies such as PCM and STT-MRAM. With the hint of heterogeneous mini-rank, we discover that by using a bridge chip on DIMM, we can build a system that the underlining memory device can be selected and configured to fit the system requirement. The advantage of new memory technologies can be utilized and the shortcomings can be avoided using the mature DDR technology. To realize this idea, future systems may have to use different memory controllers for various types of memory modules, which is infeasible in economic sense. Furthermore, future processors are likely to use integrated memory controllers, which means there would be many subtypes of processors. We propose Universal Memory Architecture (UniMA) to enable universal interoperability between all major processors and memory modules made by emerging memory technologies as well as DRAM. The UniMA is a framework of memory architecture rather than a particular implementation. It is an advanced memory organization with a new memory access protocol that offloads the scheduling of device-level operations to each memory module. As for the implementation, it extends the logic functions of the bridge chip when compared with the

heterogeneous mini-rank scheme. Such an extension does not incur significant extra cost or power consumption, and it inherits the improved scalability, performance and power efficiency from decoupled memory organizations. In this thesis, we evaluate an implementation of the UniMA framework on top of the DDRx bus protocol; so that we can compare it with existing memory system designs. Each memory module embeds a bridge chip that performs local management, e.g., for DRAM devices, to schedule device operations including precharge, activation and read/write. The memory controller may still perform memory access reordering, i.e. reorder memory requests based on some scheduling schemes, but without timing constraints. The bridge chip is an extra chip on the memory module like those on the FB-DIMM (86), Mini-Rank (96) and Decoupled DIMM (97), or the bottom chip in future stacked 3-D memory modules. Since the controller no longer has the full knowledge of each devices status, a new protocol is proposed for the communication between the controller and modules to avoid bus contentions. The controller sends generic commands such as reads or writes instead of device-specific ones to modules; a module with ready data will raise its readiness via some additional signal lines; and a token-based approach is used to grant bus ownership to one ready module and avoid bus contentions.

Our simulation results show that UniMA achieves comparable performance when compared with conventional DDR3 memory systems. On a simulated quad-core system with homogeneous DRAM devices, UniMA can improve performance by 3.1% on average (up to 4.5%) for memory-intensive workloads; and incurs an average performance loss of 1.0% (up to 1.8%) for non memory-intensive workloads. UniMA may also support heterogeneous devices in one system,

which is not feasible under the conventional memory organization. A heterogeneous system of PCM and DRAM, for example, may balance the large capacity and relatively high power efficiency of PCM with the relatively high performance of DRAM. Our simulation results show that compared with the performance of a pure DRAM system, the overall performance of a pure PCM system is about 25% lower for memory-intensive workloads, while the performance of a hybrid system enabled by our UniMA design is only 12% lower.

Throughout this thesis, we demonstrate that our schemes can save DRAM power, provide optimal energy efficiency for mini-rank kind of design and integrate diverse memory technologies into one memory system with small overhead. A combination of the techniques in this thesis is straightforward and would be able to show a path to utilize both dominant and future memory technologies in a single computer system to reduce memory power, increase capacity and performance.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Memory Organization

A conventional memory system may consist of a few memory channels. Each channel can operate independently and connect with one to four DIMMs. Each DIMM consists of several DRAM devices that are organized as ranks to serve requests. In desktop and server environment, the data bus in a channel is usually 64-bit wide and is shared by all devices connected to the channel. Using the most widely used x8 devices as an example, eight devices will form a rank to match the bus width and serve memory requests together. For a typical memory system with last-level cache block size of 64 bytes, the 64-byte data of each cache line filling request will be spread onto eight devices, with each holding 64-bit data.

2.2 Memory Technologies

2.2.1 DDRx SDRAM

DRAM is the dominant memory technology today with DDR (Double Data Rate) SDRAM being the most widely used. The DDR interface uses double pumping mechanism to transfer data on both the rising and falling edges of the clock. It has several generations, DDR, DDR2, DDR3 and with DDR4 projected.

Figure 1 shows the internal architecture of a typical DDR SDRAM device. There are usually 4 or 8 banks in a DRAM device and each has a row buffer. When a row activation command

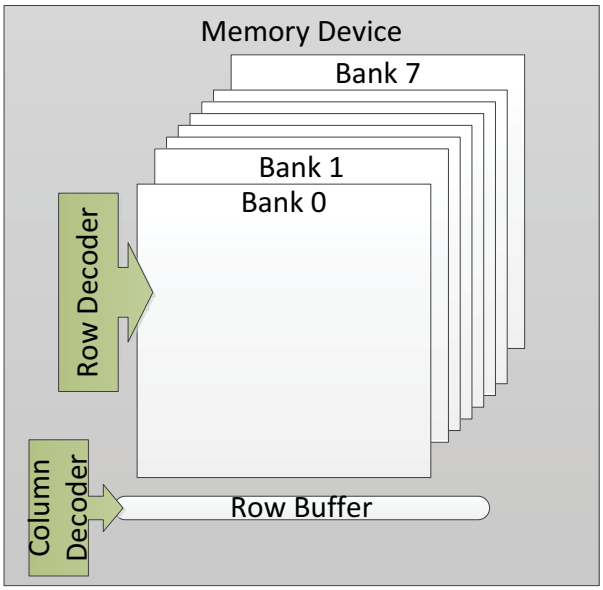


Figure 1: Basic internal architecture of DRAM device.

and corresponding row address are received, the row decoder translates the row address and selects a row from the corresponding DRAM bank. Then the contents of the selected row are fetched to the row buffer. When there are valid data in the row buffer, the bank is open to accept subsequent column access requests. A column access command can be issued to an open bank to transfer data to or from the column selected by the column decoder. If data in any row other than the one currently in the row buffer need to be accessed, a precharge command will be issued to the bank, which stores the data in row buffer back to DRAM core. After that, a new set of row activation and column access can be performed. All the read and write operations on DRAM devices are accomplished by scheduling these basic commands.

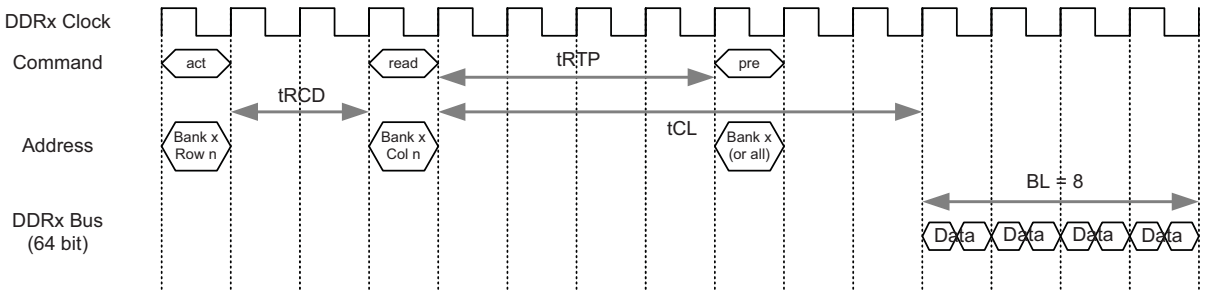


Figure 2: A simple timing diagram of DDR3-1600 (tRCD: RAS to CAS delay, tCL: CAS latency, tRTP: read to precharge time).

DRAM devices operate as slaves of a memory controller. The controller schedules concurrent memory requests and sends commands (including precharge, row and column accesses) to devices for each request based on devices' status and timing constraints. Figure 2 shows an example of the timing and status of command and data buses of a DDR3-1600 memory system.

With DDR technology, a maximum transfer rate of 1600 MB/s could be achieved with 100 MHz bus frequency (53). The superseding DDR2 and DDR3 technology achieves higher bus speed than DDR with the same memory clock since their internal clock is set at one quarter and one eighth the speed of data bus. Also DDR2 and DDR3 use advanced device technology that allows lower operation voltage and higher device density. The power is significantly reduced and the latency is also improved slightly. At the 2008 Intel Developer Forum in San Francisco, DDR4 was revealed and said to debut in 2012. Although sometimes they are called DDRx for simplicity, they are not compatible, which means different generations of DDR devices cannot be used together.

Graphics Double Data Rate 3 (GDDR3) and its successors GDDR4 and GDDR5 are specially designed on the base of DDR2 and DDR3 to handle certain graphics demands better, and are widely used on graphic cards.

The XDR (eXtreme Data Rate) DRAM is the successor to the Direct Rambus DRAM. It heavily emphasizes on per pin bandwidth and is designed for small, high-bandwidth consumer systems and high-end GPUs. It is used in Sony PlayStation 3 console. Also XDR2 DRAM was announced around mid 2005 (70).

2.2.2 DDR SDRAM Power Modes

The DRAM device can enter into several different power modes to reduce its power consumption if there is no access to the device. Figure 3 shows different power modes of a typical DDR SDRAM device and the transition between each mode. When there are one or more banks open in a rank, the rank is in the Active Standby mode. The rank can enter the Active Power-down mode by putting the CKE (clock enable) signal to low. If there is no bank open in a rank, the rank is in the Precharge Standby mode (fast or slow). When only the CKE is put to low, the rank can be put into the Precharge Power-down (fast) mode. If the DLL (delay-locked loop) is also disabled, it will be put into the Precharge Power-down (slow) mode, which consumes less power than the fast mode but requires longer time to return to the standby mode and then serve incoming requests. The rank can also be put into the Self-refresh mode that just performs the periodic refresh operation to maintain the data and consumes the least power. The current value of each mode is also shown in the figure. The data is based on the Micron 2GB DDR3-1600 x8 device (52). We can see that at the Active/Precharge Standby mode,

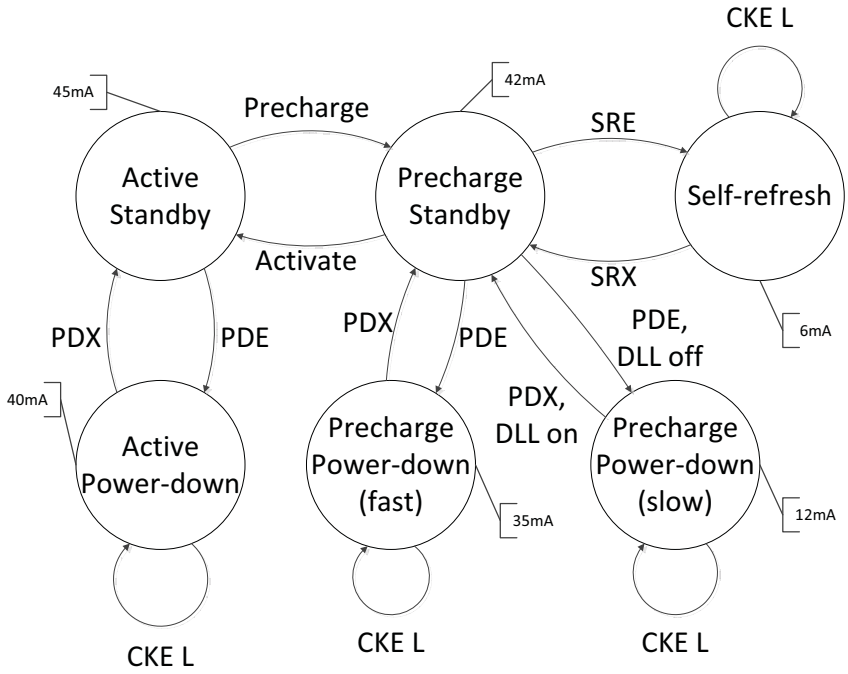


Figure 3: Low power modes for DDR3-1600 memory device and the current under each mode. PDE: Power-Down Enter; PDX: Power-Down eXit; SRE: Self-Refresh Enter; SRX: Self-Refresh eXit.

the current (determining power consumption) is higher than that at the Precharge Power-down fast/slow modes (45/42 mA vs. 35/12 mA). The power mode determines the background power. When the rank is performing an activation or a precharge operation, the operation power is consumed. When the rank is bursting data in/out, the I/O and termination power is consumed. Background power, operation power and I/O and termination power are three sources of power consumed by a rank during normal operations.

2.2.3 MRAM and PCM

Magnetic RAM (MRAM), Spin-Torque Transfer Magnetic RAM (STT-MRAM) and Phase Change Memory (PCM) are non-volatile memory technologies that have been under extensively research and are potential alternatives to DRAMs (16; 41; 69). They use the resistance of cell to represent information and are also called resistive memories (28). Unlike conventional DRAM technologies, MRAM, STT-MRAM and PCM do not store data as electrical charge. MRAM (STT-MRAM) uses the different resistance of magnetic storage elements determined by the polarity of its two layers; while PCM uses the obviously different resistance of chalcogenide glass between its two states: amorphous and crystalline, to represent data. Samsung ships industry's first Multi-chip Package with PCM on April, 2010 (74). The most basic MRAM cell size is limited to around 18 nm due to the half-select problem. Although Toggle-mode MRAM could be made into much smaller size, a worthy of putting into wide production is not shown in the near future. PCM on the other hand have their own advantages and disadvantages compared with DRAM. For instance, PCM has longer latency and worse write endurance than DRAM but promises better power-efficiency and higher density.

By carefully controlling the programming voltage and electric current, the phase change material can partially transit between phases. Thus, intermediate resistance/state can be achieved. This characteristic of PCM allows each PCM cell to have multiple states and be used to build Multi-Level Cell PCM (MLC PCM). A multi-level cell can store several bits. For example, if a MLC PCM cell has four states, it can represent two bits; and if it has sixteen states, four bits can be stored. Thus, MLC PCM can double or even quadruple the storage density. However,

the more states a MLC PCM has, the more difficult it becomes to bring a MLC PCM cell to its desired state region because the margin is smaller. So usually a write-verify type of write strategy (60) is implemented to gradually set a cell to the right state. The drawback is that such multiple write-verify iterations will make write to MLC PCM to be much slower and wear out cells much faster than single-level cell PCM.

2.2.4 Mini-Rank Architecture

In a typical DDR2/DDR3 DRAM memory system, multiple DRAM devices (chips) on a DIMM form a group to match the 64-bit data path and serve one memory request. Such a group is called a rank (e.g., sixteen x4 DRAM devices forming a x64 rank). With the fast increase on memory channel bandwidth and the slow improvement on DRAM row and column access speed, the bus transfer time only counts for a small portion of the overall memory access latency in today's memory system. By observing that fact, a recent study proposed a mini-rank architecture that partitions each memory rank into multiple mini-ranks with narrower width (e.g. a x64 rank into two x32 mini-ranks) and lets each mini-rank serve one memory request (96). It can significantly reduce the memory operation power since fewer devices are activated for each request. In addition, it can also reduce the background power because more independent groups of devices mean better chance to utilize the low-power modes. The performance penalty mainly comes from the increase on the bus transfer time for a request. However, since that only counts for a small portion of the total memory access latency, the overall performance penalty is small. In order to relay data between the wide data bus and the narrow mini-rank, a bridge

chip called mini-rank buffer (MRB), is added to each DIMM. The data relay and transfer on the bus can be pipelined and partially overlapped to minimize the overhead.

The power saving comes from better use of low power modes mentioned above, less operation power and less termination power due to the isolation effects brought by the MRB. As mini-rank breaks the original x64 rank into smaller ranks, fewer devices are involved when an operation is issued by the memory controller. Because operation power grows proportionally as the number of device decreases, smaller ranks costs less operation power than x64 rank. Using mini-rank will double the number of logic ranks in x32 configuration, quadruple in x16 and octuple in x8 configuration. Giving that the number of memory requests waiting for service almost has nothing to do with the memory subsystem architecture and unit memory operation is based on rank, when the number of rank increases, more ranks could gain the opportunity to go into low power mode. The divided mini-ranks are connected to MRB so that they do not share the same bus as in non-mini-rank design, which eliminate the need for rank termination.

2.2.5 Related Work

Many studies have been focused on how to utilize low power modes of DRAM. Lebeck et al. proposed power-aware page allocation policy to increase the chance of DRAM chips to be put in low power modes (40). Delaluz et al. proposed software and hardware approaches and OS scheduling methods to reduce memory power (15). Fan et al. studied control policies for memory power (20; 21). Huang et al. proposed a virtual memory scheme to manage memory power (29) and maximize the chip low power time by dealing with memory traffic (30). Li et al. proposed performance guaranteed low power management schemes (43). Pandey et al. extended

the memory power management aspect by considering DMA access behaviors (61). Hur and Lin studied power saving potential of memory scheduling and memory power control schemes via throttling (32). Mrinmoy et al. studied refreshment algorithms to reduce memory refresh powe (27). Diniz et al. proposed algorithms to reduce Rambus DRAM power by dynamically adjusting chip power states (18).

There have been many studies on memory system performance and power analysis. Burger et al. evaluated several techniques to hide memory access latencies (8). They discovered that one of the limitations of modern computer system design is memory bandwidth. Cuppu et al. compared several different memory architectures for their latency, bandwidth and cost tradeoffs (14). Cuppu and Jacob evaluated the performance impact of various memory design parameters (13). Ganesh et al. evaluated Fully-Buffered DIMM and DDRx (26). They analyzed the scalability of capacity and bandwidth of FB-DIMM in detail. A recent paper by Zheng et al.(98) compared different memory design parameters and their impact on memory system power consumption.

There are also studies proposed to modify memory system architecture or management policies to reduce memory power. For example, Zheng et al. proposed a mini-rank architecture that reduces memory operation power and background power by activating fewer devices for each memory request (96). Ahn et al. proposed Multicore DIMM used the rank subset concept with split data bus to reduce the memory power (2). Udipi et al. proposed to split the DRAM row buffer to reduce memory power consumption via less operation power (85), which will require significant change of commercial device. Kaseridis et al. proposed DRAM page policy

that sets the target of page hit number and schedules memory requests accordingly (37) to optimize page open time and reduce activation power. Lim et al. proposed a new memory blade architecture for blade servers. They used a dedicated blade for shared memory storage and a stand-alone memory control facility is given to each memory blade (44). Beamer et al. proposed a new memory system bus architecture using Monolithically Integrated Silicon Photonics. The bus speed increases significantly and an inhibited memory scalability is empowered (5). Several other studies focused on using low-power DRAM to build low power memory systems (90; 46).

Many studies proposed to use a bridge chip on DIMM for the purpose of performance-endurance and storage tradeoff. Registered DIMM (54) puts a register chip on each DIMM to cache command/address from memory controller. So the electrical load is reduced for supporting more DIMMs. MetaRAM (51) not only buffers command and address but also data to reduce the number of visible ranks to memory controller, which also reduces the load on buses. Fully-Buffered DIMM (35) relay multiple DIMMs via Advanced Memory Buffer that more DIMMs could be put into one channel. Also the narrowed buses made it possible to put more channels on the motheboard. Decoupled DIMM (97) uses a sync-buffer to bridge the speed differences of on-DIMM devices and DDRx data bus so high speed bus could be used by low speed memory devices.

There have also been many other studies on memory scheduling algorithms and performance analysis. There were mapping schemes improve the row page hit rate and reduce the latency (95; 45). McKee and Wulf study the effectiveness of five access ordering schemes on a uniprocessor system (49). They also studied them on SMP systems for vector-like stream accesses (50).

The Impulse memory controller supports application-specific optimizations through address translation (remapping) to improve bus and cache utilization, and also supports prefetching at the memory controller to hide the cost of remapping (9; 93). Hur and Lin propose adaptive history-based scheduling policies to minimize the expected delay and match the program's mixture of reads and writes (31). Zhu and Zhang evaluate memory optimizations for the SMT processors and propose thread-aware scheduling schemes based on the pending request number and processor resource usage (100). Jun and Brian propose a burst scheduling mechanism to cluster the accesses on the same row page to maximize the data bus utilization (78). Mathew et al. design a Parallel Vector Access Unit for the Impulse controller, which gathers sparse and strided data structures to improve memory locality and reduce observed memory latencies (47). Fine-grain scheduling schemes can improve the performance of multi-channel memory systems (101). Rixner analyzes the effects of policies utilizing channel buffers of virtual channel SDRAM and memory access scheduling schemes in reducing memory access latency in web servers (72). Mckee et al. propose to reorder memory requests (48) to reduce memory read latency. Rixner et al. proposed the FR-FCFS scheduler (73) to improve memory throughput. Nesbit et al. used fair queueing to schedule memory requests for fairness and QoS (59). STFM (57) estimated the slowdown of each thread and prioritizes the thread that had been slowed down the most. PAR-BS (58) formed a batch of requests for threads and prioritized the one with fewest requests for performance purpose. The ATLAS scheduler (38), aimed to maximize system throughput by prioritizing threads that have attained the least service. TCM (39) put threads in two groups via memory intensiveness and schedule them to get the optimal

fairness and throughput. The Prefetch-Aware memory scheduler (19), identified the prefetch streams and scheduled non-prefetch and accurate prefetches ahead to improve performance. Ausavarungnirun et al. proposed a staged memory scheduler (4) to co-schedule CPU and GPU accesses.

There have also been many recent studies on issues related to using PCM and MRAM memory technologies, such as analyzing the scaling behavior of PCM (71), using PCM as the substitute of DRAM (41; 69), improving write endurance of PCM (99; 68), and using PCM to improve cache organizations (88; 82). Other studies include write cancellation and pausing to improve PCM performance (66), using PCM as the flash memory log for better efficiency and lifetime (83), using non-volatile memory to build high performance data arrays (10), and flipping the data bits to save power (11). There are also studies regarding write strategies (60; 36; 94), dynamically adjusting MLC PCM for optimal latency-capacity trade-off (67), a demo of MLC PCM (62), and changing the cell encoding for lower power (89). Some use STT-MRAM blocks to reduce leakage power of processor (28), and using on chip interconnections to reroute accesses to STT-RAM (55).

There are also research studies on the impact of putting different memory technologies into one system, such as the cache hierarchy to form a hybrid cache (88), evaluation on a hybrid DRAM and PCM system and guidelines for possible designs (7), hybrid memory system for low power mobile applications (42), using PDRAM memory to save memory system energy and manage PCM wear leveling (17), a MLP aware heterogeneous memory using off-line profiling method to map applications to proper memory regions (65), OS supports for hybrid memory

management (56), and a architecture of hybrid SDRAM and STT-RAM cache that relaxes the STT-RAM design to reduce energy-delay product (12).

CHAPTER 3

CONSERVATIVE ROW ACTIVATION TO SAVE MEMORY POWER

3.1 Introduction

In recent years, the memory bandwidth and capacity has been improved steadily to meet the user demands on higher bandwidth and larger capacity. A negative side-effect is the continuous increase on memory power consumption, which has become a first-order design consideration for memory systems. In this study, we propose a new scheme to reduce memory power consumption with negligible performance impact.

As we enter the era of multi-core systems, memory-level parallelism has become a very important performance factor. This is because multiple concurrently running processes generate multiple memory access streams and they tend to destroy locality appeared at the memory side. To support multiple concurrent requests, a typical memory system has multiple ranks and banks, which can serve requests in parallel. The memory controller will schedule device and bus transactions for those concurrent requests to meet timing constraints and avoid resource contentions. A common strategy used by today's controllers is, whenever possible, always trying to open a memory bank for a pending request to maximize bank-level parallelism and thus throughput. However, we find that this is neither power optimal nor necessary for maintaining good performance. Since memory requests tend to cluster together, it is common that several

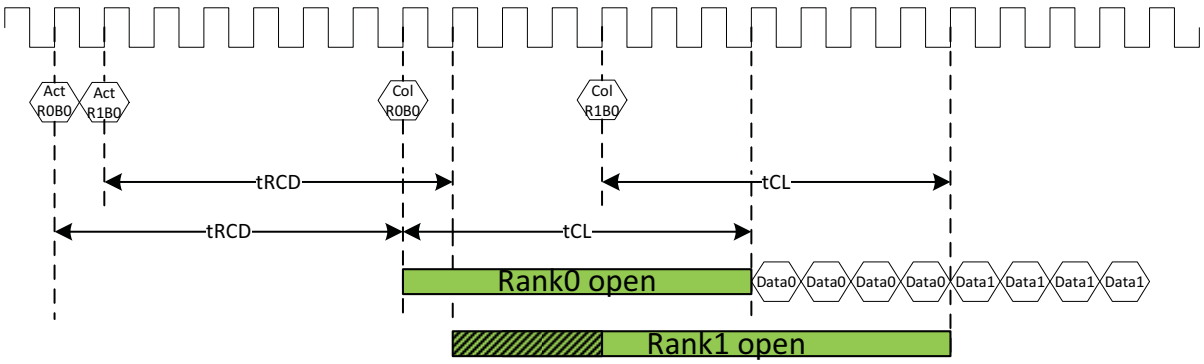


Figure 4: Conventional timing diagram of two concurrent requests going to different ranks. (The shaded bar represents the wasted cycles due to data bus contention.)

banks are open but most of them are just stay idle, waiting for the data bus to become free. That wastes unnecessary power and contributes little to extra performance.

Figure 4 shows an example of two concurrent read requests going to two banks in different ranks. Each request has an activation command followed by a column access command. Once an activation command is sent out, after a delay of tRCD, the corresponding rank enters the active state, the bank is open, and the row containing the required data is fetched to the row buffer. The first request (mapped to Rank0 in the example) can immediately issue its column access command once its bank is open because the data bus is free at that time. However, the second request (mapped to Rank1 in the example) has to wait three more bus cycles for the data bus becoming free after its bank is open and ready to serve column accesses. This is because the data bus is shared among all the ranks connected to a channel and can only serve one data transfer at a time. The extra three cycles of Rank1 in the high-power, active state is not necessary for delivering the maximal performance. If the activation of Rank1 is delayed

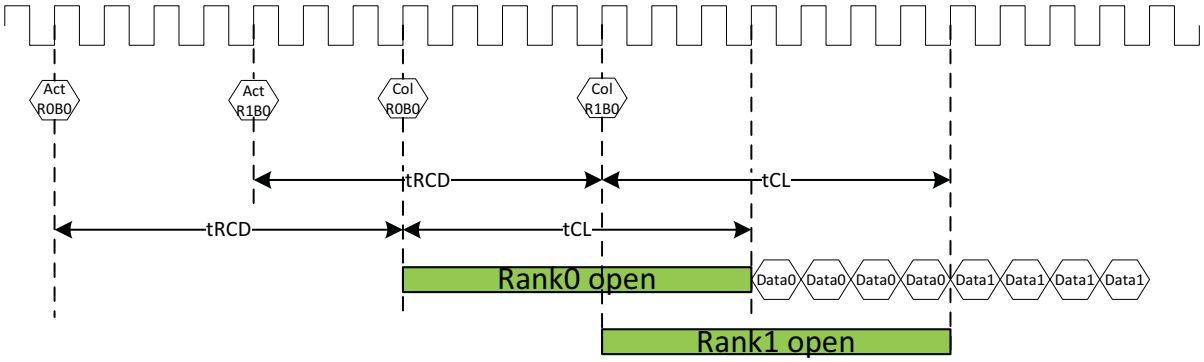


Figure 5: Timing diagram of Conservative Row Activation for two concurrent requests going to different ranks.

by three cycles as shown in Figure 5, the second request can issue its column access command right after a delay of tRCD once its activation command is issued, and there are no cycles for Rank1 to be wasted in the high-power, active state. This example shows delaying bank/row activation until its corresponding column access not being blocked by the busy data bus can reduce the time that a rank stays at the high-power, active state to save memory power without compromising performance.

There are two questions to be answered before the idea of delaying row activation can be applied in real systems to improve memory power efficiency. The first one is whether the optimal delay time of a row activation can be determined efficiently. The answer to this question is yes. Because the memory controller knows the status and timing of each bank and each pending request, and it also knows when the command and data buses become free. Thus, for a given request, the controller knows the earliest time that the column access command can be sent out. It can use that information to calculate backward the latest time when the row activation

command should be sent out, since the row activation and column access commands must be separated by at least a time of tRCD.

The second question is whether it happens frequently in real systems that several banks are open but idle waiting for data bus becoming free. We further verify with experiments that such situation does happen frequently in real systems. We collect the statistics of the number of cycles that a bank is in active state but cannot issue its column command because the data bus is not free. Then we calculate the metric of "possibility that a bank open but waiting for free data bus". The metric can give us an idea of how often that a bank stays at the unnecessary open state wasting energy. For instance, we find out that for a memory-intensive workload consisting of SPEC2006 applications lbm, gemsFDTD, leslie3d and libquantum, the experimental result on a dual-channel configuration with two DIMMs per channel and two ranks per DIMM shows that on average a bank wastes 11.5% of time in active state but waiting for a free data bus to issue column commands. Note that a rank usually contains four or eight banks. So a rank will be more likely to stay at the active state but the data bus is used by other ranks.

Based on those analysis and observations, we propose a "Conservative Row Activation" memory access scheduling scheme to improve memory system power efficiency with minimal performance impact. Instead of activating a bank at the earliest possible time, it activates a bank only when the follow-up column access will not wait for the data bus ownership. The scheme monitors the data bus transactions and reserves bus slots for each pending column command at the earliest possible time. Then the optimal time to issue the corresponding

row activation is calculated as tRCD cycles before the column command. In this way, more banks can stay in idle state longer and ranks will get a better chance to stay in the low-power, precharge state to save power. In addition, because the column commands can still be issued almost at the same time as before, the performance impact would be minimal.

We build a detailed memory model and integrate it to the MARSS (63) simulator. Running the quad-core workloads with SPEC2006 (81) benchmark, the experiment indicates that our proposed scheme can effectively improve memory system power efficiency with little performance impact. The memory power is saved by 5.6%, 3.3% and 5.8% on average for MEM, ILP and MIX workloads, respectively (up to 7.0%). The performance is even slightly improved for MEM workloads because our scheme can slightly improve the bus utilization by giving column access commands higher priority than others. The weighted speedup for MEM workloads is improved by 0.3% on average (up to 1.0%). For ILP and MIX workloads, the performance loss is less than 1.0%.

3.2 Conservative Row Activation

As described in Section 3.1, conventional memory schedulers usually open a row at the earliest possible time to maximize memory-level parallelism and to minimize memory access latency. However, we find that although such scheduling policy is simple and performance-optimal, it wastes unnecessary energy. For today's memory systems, it is common that several requests are served by the memory system at the same time since multiple threads/applications are running concurrently on the multi-core processors. We find that when there exist multiple

concurrent memory requests, some of the row activation operations can be delayed to save memory power without performance penalty.

In a typical desktop/server system with DDRx memory system, the memory bus is eight-byte wide and the block size of the last level cache is 64 bytes. Each cache miss request from the last level cache will fetch 64-byte data in a burst, whose length is eight. Thus, each burst occupy the DDRx data bus for four bus cycles to transfer its data. Because each memory bus/channel can only transfer one burst at a time, all data transfers must be serialized by the bus. In general, the data bus is the performance bottleneck instead of the command/address bus. A typical memory request occupies the data bus for four cycles (with additional cycles for read/write turn-around), while its row activation and column access commands each occupies the command bus for one cycle. Thus, when the row activation commands of two concurrent requests to different banks are sent out back to back, the bank to serve the second request will sit in the open state waiting for the data bus becoming free to perform its column access. Such waiting in the high-power open state is unnecessary. It consumes extra power but contributes little to performance.

To address this issue, we propose a memory scheduling scheme called "**Conservative Row Activation**". It delays the row activation of a pending request until its subsequent column access does not need to wait for the data bus becoming free. This allows memory ranks to stay in the low-power idle state longer to save memory power. At the same time, our scheme tries to avoid delaying column accesses to minimize the performance impact.

3.2.1 Basic Scheme

Memory system performance should always be considered at the highest priority because it directly affects the overall system performance. Bear this in mind, we propose the "Conservative Row Activation" memory scheduling scheme to save memory power with minimal performance impact. To maximize the rank idle time for memory power saving, our scheme delays the row activation command as late as possible. However, a row has to be opened for column access to get the data transferred. Moreover, a column cannot be immediately read from or written to after the row activation command, a delay of tRCD (row access to column access delay) must be expired between the row activation and its column access. Thus, to minimize the performance impact, our scheme should avoid delaying the column access when the row access is delayed. In order to achieve the goal, our scheme would carefully calculate and then schedule the row command so that a bank will be opened at the latest time without delaying the column command.

Algorithm 1 describes the scheduling of row activation and column access commands by our proposed scheme. The basic idea is to look ahead for data bus availability and reserve the time slot on the bus for a pending column access. Once the time slot of the column access is determined, the timing of its associated row activation can be calculated using the value of tRCD, and the activation command can be scheduled accordingly. When a pending request goes to a bank, whether an activation can be issued must satisfy the following constraint: *Its column command can be issued immediately after the bank enters the open state.*

```
for Every memory cycle at best available bank do
    if This cycle is reserved then
        Issue the column command;
    else
        if Bank Idle then
            Check timing for activation and column access at tRCD memory cycles later;
            if Success then: Reserve cmd bus for column access at tRCD memory cycles
            later AND Issue activation for this bank;
        else
            Check timing for column access at tRCD memory cycles later for row hits;
            And reserve bus slots if available;
end
```

Algorithm 1: Conservative Row Activation algorithm.

To ensure this, the scheduler looks ahead for tRCD memory cycles and tries to reserve command bus slots for the column command first. If there is no timing conflicts, the memory controller will mark the command bus for this request's column access at tRCD cycles in the future and issue the row activation at this cycle. Otherwise, the row activation will not be issued now, and such procedure will repeat the next cycle. In this way, most of the banks can be put at idle state and let the rank enter precharge standby low-power mode to save memory power. For row-buffer hits, the memory controller will also look ahead for tRCD memory cycles and reserve the slot for its column access. This is because the looking ahead and reserving scheme implicitly prioritizes reserved column accesses. The row-buffer hits should be given the same privilege to acquire the data bus as the row-buffer misses; otherwise, they will be starved.

To avoid modification on the memory device and DDR command and also ensure the correctness of memory access scheduling, a reserved column access must be issued at its designated cycle and cannot be issued on any other time. Thus, at each memory cycle, the memory controller will first check if the current cycle is reserved for any column access. If so, the reserved column command is issued. Otherwise, this cycle can be used to test a new activation and make reservation for its column command.

3.2.2 Implementation

The hardware implementation of our scheme in the memory controller is simple. An array called "Bus Reservation Table" (BRT) is added for each memory channel to record the reserved command bus slots for pending column accesses. The data bus reservation does not need to be recorded because the timing of a data bus transfer is determined once its corresponding column access command is issued. For DDRx memory devices, once a column access command is issued, depending on whether the request is a read or a write, after a delay of tRL (column command to read data delay) or tWL (column command to write data delay) cycles, the data bus will be occupied for four memory cycles by the request.

Each entry of the BRT (Bus Reservation Table) will record a request ID for whom the time slot has been reserved. It also contains a valid bit to tell the memory controller if the entry is a valid reserve or the entry should be ignored. The BRT is referenced as a circular queue and a pointer is used to indicate which entry of BRT stands for the current cycle. The pointer will increase by one every memory cycle, and it will be wrapped around and pointing to the BRT head when reaching the tail of BRT.

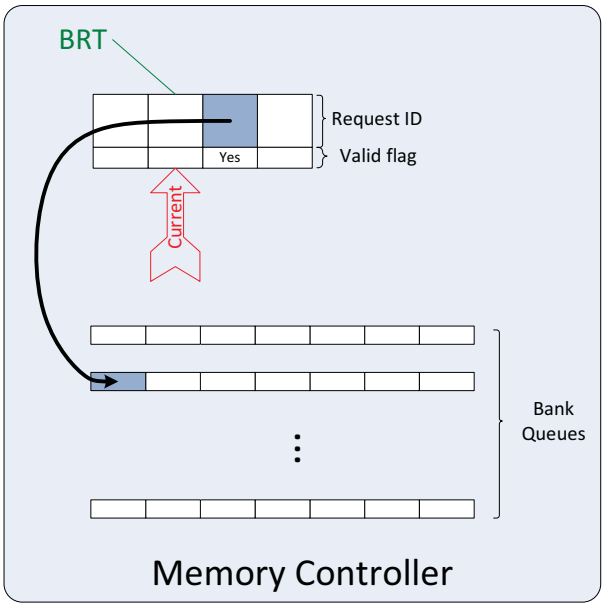


Figure 6: BRT and bank queues in memory controller. The reserved BRT entry will point to the request in bank queue.

Figure 6 shows the structure of the BRT and the bank queues in the memory controller. The bank queues are structures in the memory controller that buffer and schedule requests for each bank. As the simple example shown in the figure, a BRT slot is reserved for a column access (its associated activation has already been served). The *current* pointer points to one slot ahead of the reserved slot, which means a column access is reserved for the next memory cycle. Then at the next memory cycle, the *current* pointer will point to the reserved entry. The valid flag is set for the entry. Thus, the entry is valid and the Request ID field will tell the memory controller to find the corresponding request in the memory bank queues and issue corresponding command for it.

3.2.3 Hardware Overhead

As discussed in Section 3.2.2, the additional hardware to support the Conservative Row Activation scheme is BRT. Every time when a reservation is made, the memory controller will look ahead for tRCD cycles and try to reserve a command bus slot there. So the minimum number of entries for BRT is tRCD + 1 (11 + 1 for DDR3-1600 memory device). The maximum number of entries should depend on how many slots that it would like to reserve in the future. Because the memory controller can update the BRT on every memory cycle, there is no need to reserve more than two requests. Given that each request is separated by four or five cycles if they satisfy the requirement stated in Section 3.2.1, the maximum number of BRT entries is 32. The valid flag will need one bit. The Request ID can either be the request number if the memory controller has a central facility to keep tracking every pending memory requests or the rank and bank index of each request otherwise. Considering a typical memory system today, the estimated size of one BRT entry is no more than six bits. So the total size of BRT per channel would be 192 bits.

The memory controller should remember the reserved requests so that it will not reschedule a reserved request. A rescheduling may change the bank state and waste the reserved time slot. Thus, we propose to add a one-bit "reserved" flag for each entry of the bank queues. The additional storage overhead will be within several hundred bits depending on the size of the bank queues for a memory controller implementation. In summary, the hardware cost to implement our scheduling algorithm is trivial.

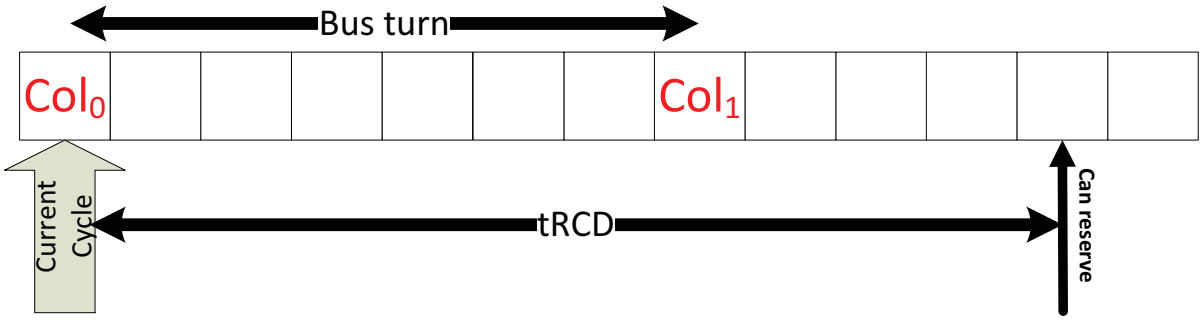


Figure 7: An example of row and column command conflicts at the same cycle.

3.2.4 Improvements from Basic Scheme

The basic Conservative Row Activation scheme is simple but strictly requires a column access be immediately scheduled after its bank is open, which sometimes introduces conflicts on command bus and thus decreases the command bus utilization and performance. Figure 7 shows a simple example of conflicts on the command bus.

As shown in the figure, Col_0 and Col_1 are previously reserved command bus slots for column accesses to Bank 0 and Bank 1, respectively. Col_0 and Col_1 are at the turning point from a read burst to a write burst. Due to bus turn-around delay, they are apart by seven memory cycles. The closest next available time slot for a new column access is at Cycle 11. However, the cycle cannot be reserved for a new column access because that would require an activation to be issued at the current cycle to open the row. However, the current cycle has already been reserved for the column access to Bank 0. As a result, the memory controller will choose to issue the column command for Bank 0 and delay the activation for the new request to the next

cycle, and reserve Cycle 12 for its column access. There will be a bubble on the data bus, which reduces the data bus utilization. The schedule of DRAM commands is very dynamic and complex. There will be many cases that commands may conflict with each other and cause bubbles on the data bus. This will reduce memory system performance.

To resolve the problem, we modify the original scheme and relax a little bit on the constraint stated in Section 3.2.1 to: *The column command can be issued immediately or one memory cycle after the bank enters the open state.*

The relaxed rule allows a bank to stay in the open state for one more memory cycle. The modification gives the activation command a two-memory-cycle window to minimize the performance loss. Choosing a window wider than two memory cycles may further reduce command bus conflicts. However, that will reduce memory power saving and complicate the scheduling algorithm. From experiments, we find that the two-cycle window is a good design choice.

Another optimization is done to further reduce memory power. The scheme saves memory power mainly by reducing the time that each rank stays at the active state. If requests are spread across several ranks, it is better to schedule requests in the same rank together and then go to the next rank instead of jumping across ranks frequently. Thus, our revised scheme will give higher priority to requests going to the current rank than those to different ranks.

Algorithm 2 shows the improved Conservative Row Activation scheme. Compared with the basic scheme, the revised one can improve the data bus utilization and thus memory performance. The side effect is that a rank may stay in the high-power open state slightly longer.

```
for Every memory cycle at best available bank in the same rank; Then different rank do
    if This cycle is reserved then
        Issue the column command;
    else
        if Bank Idle then
            Check timing for activation and column access at tRCD memory cycles later;
            (Relax 1 memory cycle if fails);
            if Success then: Reserve cmd bus and data bus for column access at tRCD
            memory cycles later AND Issue activation for this bank;
        else
            Check timing for column access at tRCD memory cycles later for row hits;
            And reserve bus slots if available;
end
```

Algorithm 2: Modified Conservative Row Activation algorithm that allows a two-cycle window for activation command to avoid timing conflict.

We also want to point out that our scheme works on a page policy that auto-precharges a row if there is no pending request on it. It is also the most widely used page policies in today's computer systems. With multi-core processors, the locality among memory access streams is much lower than that with a single memory access stream. The page policy we use tends to perform better than open page policy.

For the open page policy, it lets the bank stay in active state until it must be precharged to serve a conflict memory request, our scheme would have smaller impact on memory power and performance because memory ranks will have little chance to be at precharge low-power state.

TABLE I: Major simulation parameters for CRA.

Parameter	Value
Processor	4core, 3.2GHz, 4-issue per core, 14-stage pipeline
Functional units	2 IntALU, 4 LSU, 2 FPALU,
IQ, ROB and LSQ size	IQ 32, ROB 128, LQ 48, SQ 44
Physical register num	128 Int, 128 FP, 128 BR, 128 ST
Branch prediction	Combined, 6k bimodal + 6k two level, 1K RAS, 4k-entry and 4-way BTB
L1 caches (per core)	64KB Inst/64KB Data, 2-way, 16B line, hit latency: 3-cycle Inst/3-cycle Data
L2 cache (shared)	4MB, 8-way, 64B line, 13-cycle hit latency
Memory	2 channels, 2 DIMMs/channel, 2 ranks/DIMM, 8 banks/rank
Memory controller	64-entry buffer, 15ns overhead
DDR3 DRAM latency	DDR3-1600:11-11-11: precharge/ row access/ column access: 13.75ns

3.3 Experimental Methodologies

We have built a detailed DDR3 DRAM memory model and integrated it into MARSSx86 (63) simulator. The simulator is capable of keeping track of each request and the state of every memory channel, rank and bank. The simulated memory controller will issue commands according to the current memory status and pending requests. The memory transactions are pipelined whenever possible. Also the XOR-based mapping (95) is used. Table I shows the major simulation parameters.

TABLE II: Parameters used for calculating DRAM Power (2GB, x8, DDR3-1600, 11-11-11).

Parameters	Values
Normal voltage	1.5V
Active precharge current	95mA
Precharge power-down standby current (fast)	35mA
Precharge power-down standby current (slow)	12mA
Precharge standby current	42mA
Active power-down standby current	40mA
Active standby current	45mA
Read burst current	180mA
Write burst current	185mA
Burst refresh current	215mA

3.3.1 Memory Power Calculation and Performance Metrics

We follow the Micron power calculation methodology (52) to estimate the power consumption of DDR3 DRAM devices. At the end of every memory cycle, the energy consumption of the current cycle is calculated based on the state of each device, and the result is recorded and accumulated. The parameters used for calculating the DRAM power are listed in Table II. For cases that the electrical current values presented in data-sheet are from the maximum device voltage, they are de-rated by the normal voltage(52).

The performance is characterized using weighted speedup (80) $\sum_{i=1}^{n}(IPC_{multi}[i]/IPC_{single}[i])$, where n is the total number of applications running, IPC_{multi}[i] is the IPC value of the application running with other instances and IPC_{single}[i] is the IPC value of the same application running alone.

TABLE III: Experimenting Workloads for CRA.

Workloads	Applications
MEM-1	lbm,gemsFDTD,leslie3d,libquantum
MEM-2	bwaves,soplex,sphinx3,cactusADM
MEM-3	mcf,perlbench,zeusmp,milc
MEM-4	libquantum,soplex,lbm,milc
ILP-1	gamess,namd,povray,calculix
ILP-2	tonto,gromacs,astar,omnetpp
ILP-3	sjeng,hmmer,gobmk,xalancbmk
ILP-4	gcc,bzip2,namd,gamess
MIX-1	lbm,libquantum,povray,calculix
MIX-2	leslie3d,gemsFDTD,gamess,namd
MIX-3	bwaves,soplex,tonto,astar
MIX-4	sphinx3,cactusADM,gromacs,omnetpp
MIX-5	mcf,perlbench,sjeng,hmmer
MIX-6	zeusmp,milc,gcc,gobmk

3.3.2 Workloads

Our experiments simulate a quad-core x86-64 system with Linux 2.6.31.4 (64-bit) running on it. We use applications from SPEC2006 (81) benchmarks to form quad-core multiprogramming workloads for evaluation and pin each application in a workload to a processor core. We use the same method described in a previous work (38) to group SPEC benchmarks into two categories: MEM (memory-intensive) and ILP (compute-intensive) based on their L2 cache misses per 1000 instructions (L2 MPKI). The MEM applications are those with larger than ten L2 MPKI and the ILP applications are those with smaller than ten L2 MPKI. Table III shows fourteen quad-core multi-programming workloads randomly formed based on their categories. The MEM-x workloads contain applications from the MEM group and the ILP-x workloads are from the

ILP group. We then pickup two applications from each group to form the MIX-x workloads to simulate workloads with different memory behavior running in the same system.

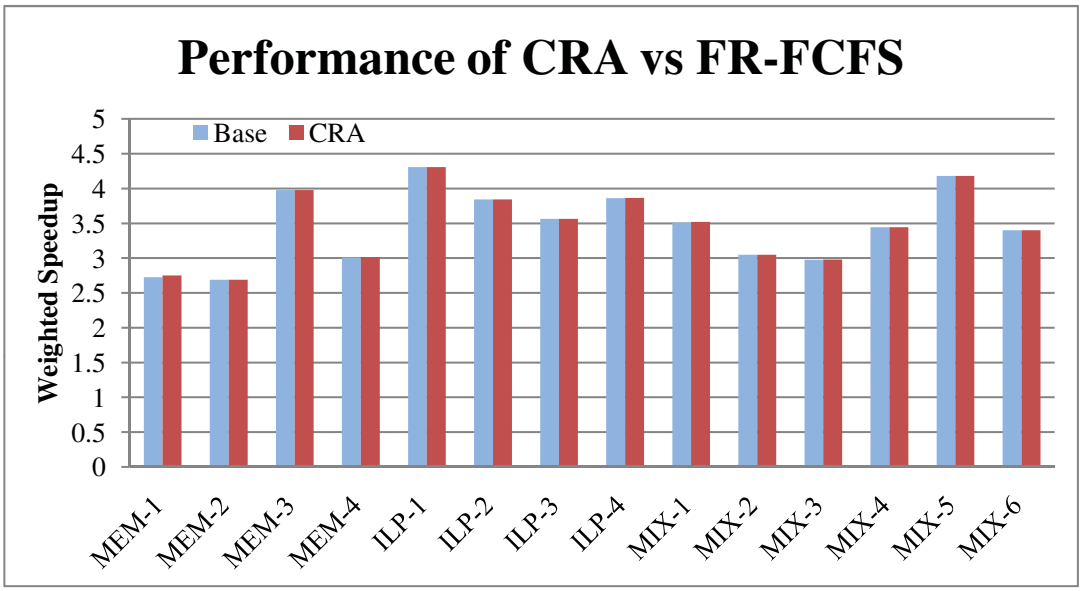
We insert a signal at the starting point of every application's main execution phase and let the application run in the fast forwarding mode until the signal is reached and then let the application sleep. When all the applications in a workload reach their signals, all the applications will be waken up and the detailed simulation and statistic collecting will launch for the following 400 million instructions.

3.4 Experimental Results

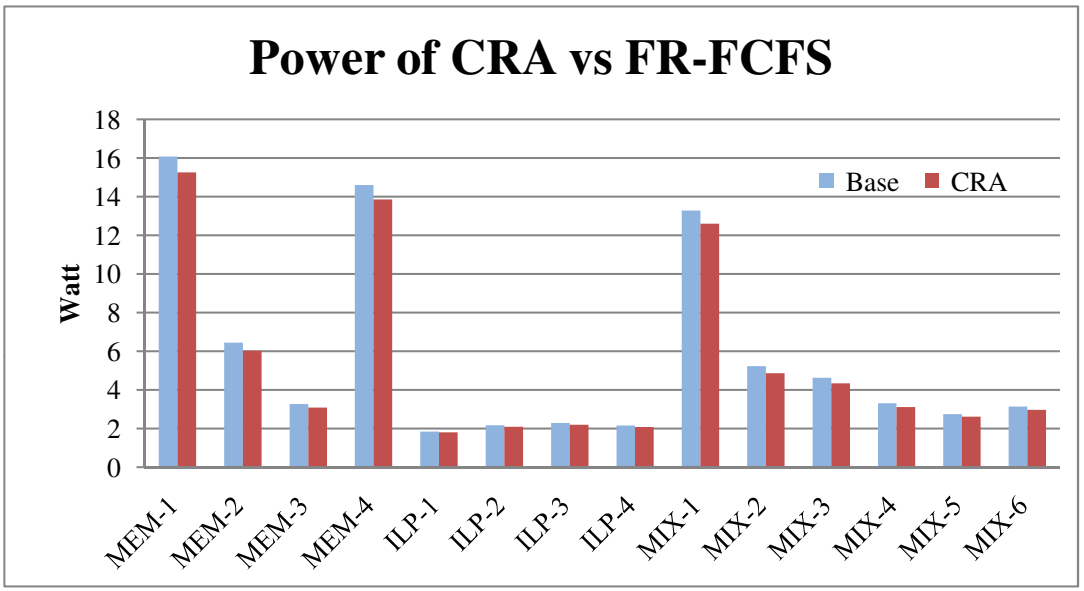
We compare our proposed Conservative Row Activation scheduling scheme with a baseline, a modified version of FR-FCFS (73) (First Read, First Come First Serve) scheduler that groups bank hits together and prioritizes reads over writes, to evaluate the performance and power consumption. Our scheme focuses on the bank activation policy and can be applied with other schedulers targeting fairness or QoS. If not specified, all the results are based on the memory configuration of two Channels, two DIMMs per Channel and two Ranks per DIMM. We use CRA to stand for our Conservative Row Activation scheme in the figures to reduce title space.

3.4.1 Overview of Performance and Power Consumption

The proposed scheme can reduce memory power by allowing a rank to stay longer in the precharged state and gives the rank more chances to be put into the low-power mode. Figure 8 compares the performance and memory power consumption of our Conservative Row Activation scheme and the baseline FR-FCFS scheme. From Figure 8a, we can see that in general, the proposed scheme achieves the same performance as the baseline. For MEM workloads, the



(a) Weighted speedup of Conservative Row Activation compared to baseline.



(b) Power consumption of Conservative Row Activation compared to baseline.

Figure 8: Overview of power saving and performance impact of Conservative Row Activation.

scheme improves performance by 0.3% on average. Especially for MEM-1, which is the most memory-intensive workload, our scheme outperforms the baseline by 1.0%. For CPU-intensive workloads, our scheme and the baseline almost perform the same. The performance difference is less than 0.09% (our scheme is slightly better). For MIX workloads, we find that for three of them (MIX-2, MIX-4 and MIX-5), our scheme performs worse than baseline. But the difference is negligible (less than 0.02%). This result shows that our scheduling scheme can retain the performance effectively. This is because although the activation is delayed, the scheme looked into the future to place column commands wisely and the latency of a request is determined by when its column access is performed not when its row activation is performed. This mechanism does not change the baseline scheduling sequence of data transfer hence the performance is kept the same.

Figure 8b shows the memory power consumption of our scheme and the baseline. The average power reduction of using our scheme is 5.6%, 3.3% and 5.8% for MEM, ILP and MIX workloads, respectively. For ILP workloads, the power saving is from 2.1% to 4.1%. This is because for CPU-intensive workloads, the memory traffic is low. The ranks are in the low-power mode for most of the time. Their power consumption ranges from 1.8W to 2.3W, which is far less than that of memory-intensive workloads (16.1W for MEM-1). When ILP workloads are running, concurrent memory requests are few, and thus the chance of banks open but data bus is not available to transfer data is low. Thus, our scheme only has small impact on ILP workloads. For memory-intensive workloads, the power reductions are from 5.1% to 6.4%. Memory-intensive workloads tend to have more concurrent memory requests. Because

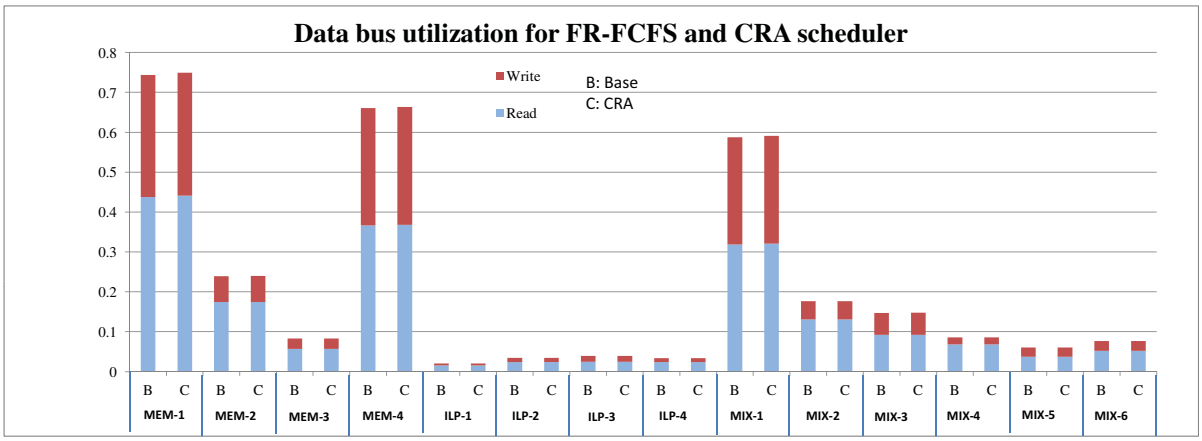


Figure 9: Data bus utilization of Conservative Row Activation compared to baseline.

conventional memory schedulers tries to open as many banks as possible, the possibility of several banks open but data bus being occupied is much higher. Our scheme can prevent banks open unnecessarily early and avoid consuming unnecessary extra power. MIX workloads have medium memory bandwidth requirement. The memory-intensive applications in them generate request concurrency that tends to open many banks at the same time, while the computation-intensive applications put less stress on the memory system. Our scheme can help saving memory power for the concurrent requests, while the total memory power consumption is less than the pure memory-intensive workloads. As a result, their power saving ratio is slightly higher than MEM workloads (from 4.7% to 7.0%).

3.4.2 Performance Analysis

To further analyze the performance impact of our scheme, we collect the memory system statistics to get an insight of why our scheme can slightly improve performance. Figure 9

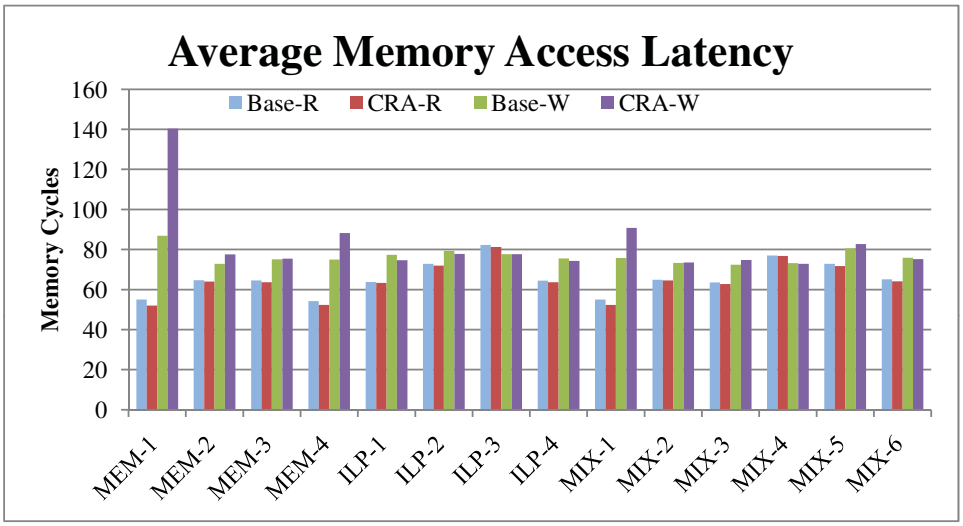


Figure 10: Memory read and write latency for Conservative Row Activation compared to baseline.

shows the utilization rate of read and write requests on the data bus. As shown in the figure, the scheme can slightly improve the data bus utilization. The biggest improvements are on workloads MEM-1, MEM-4, MIX-1 and MIX-3. The data bus utilization of read and write combined are improved by 0.8%, 0.4%, 0.6% and 0.4%, respectively. The reason behind is that our scheme reserves command and data bus slots in advance, which gives higher priority to column accesses, and the memory scheduler can make a better decision on scheduling bus transactions. In this way, our scheme can put the data transfers closer to each other and reduce the bubbles on the data bus. For the other workloads, the data bus utilization of our scheme and the baseline is almost the same (the difference is less than 0.1%). The result partly explains why using our scheme performs better than the baseline. Giving higher priority to column

accesses can increase the data bus utilization and hence the memory throughput is improved. For memory-intensive workloads, the data bus is the bottleneck. The better the data bus is utilized, the less the CPU will be blocked by outstanding load instructions.

Figure 10 shows the average read and write latency of each workload. We can see that the average read latency is reduced for all of the workloads. For MEM workloads, the read latency is reduced by 2.9% on average. Especially for MEM-1, the most memory-intensive workload, the read latency is reduced by 5.6%. The read requests are on the critical path and directly affect the system performance. The 5.6% of latency reduction leads to the 1.0% of overall performance improvement for this workload. The write latency of some workloads is increased because our proposed algorithm gives column access higher priority than other DRAM commands and it tries to issue all the outstanding reads before writes to minimize bus turn-arounds, which delays writes more. However, since the overall performance only directly depends on read latency, a longer write latency in exchange for a shorter read latency is good for performance. For the other workloads, the change of read and write latency is less obvious. The read latency is reduced by 1.1% and 1.7% for ILP and MIX workloads on average, respectively. For average write latency, the MIX workloads get a 4.1% increase on average, and the ILP workloads have a 1.8% decrease on average. Unlike MEM workloads, ILP workloads have lower memory bandwidth. Our scheme can place the transactions on data bus more compactly and thus both the read and write latency can be reduced for those workloads.

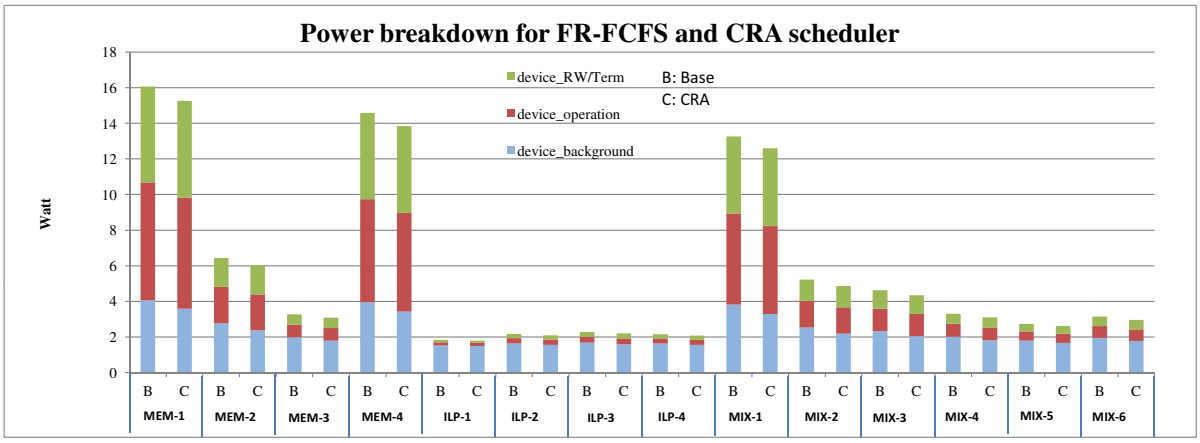


Figure 11: Power breakdown of Conservative Row Activation compared to baseline.

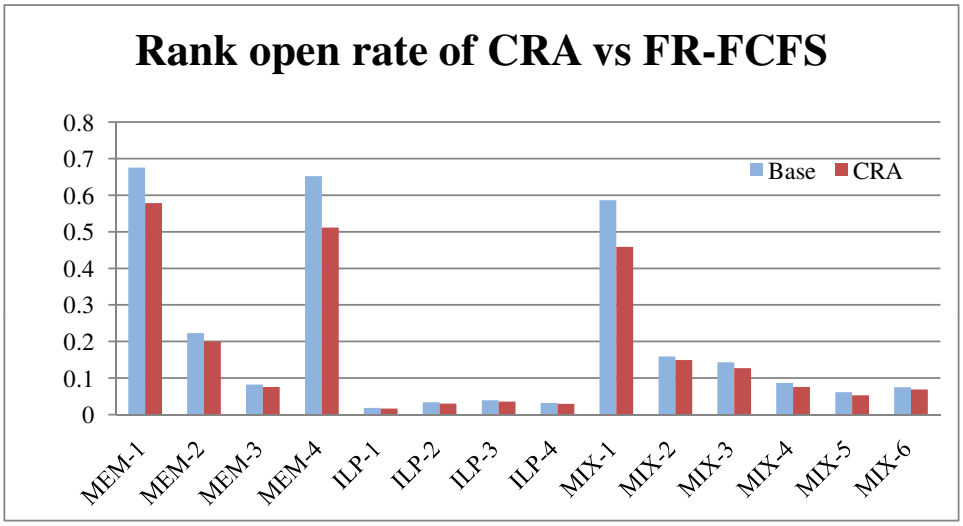


Figure 12: Percentage of average rank active time of Conservative Row Activation compared to baseline.

3.4.3 Power Analysis

We will study the source of power saving of our Conservative Row Activation scheme in this section. Figure 11 breaks down the power consumption of our experiments into three parts as in Micron DDR3 power calculator (52) and as discussed in Section 2.2.2:

1. Background power.
2. Operation power.
3. I/O and termination power.

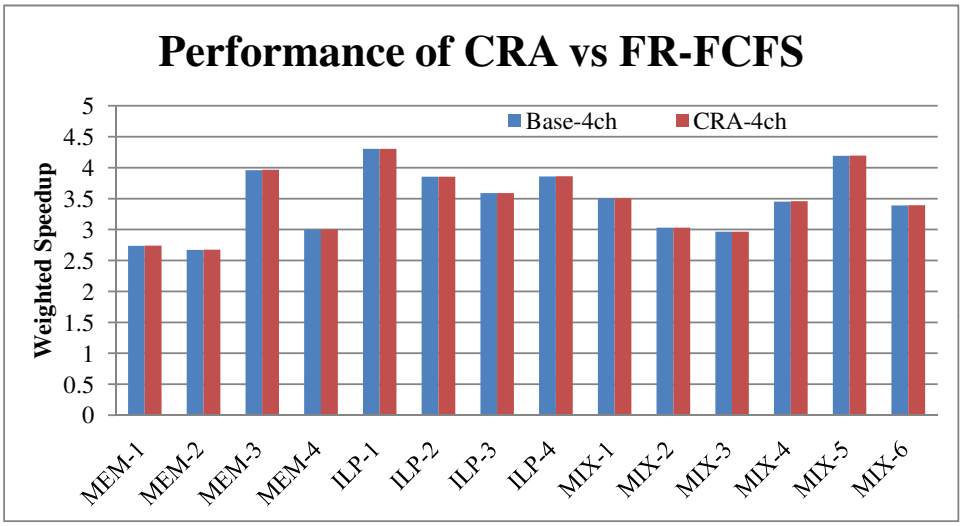
The figure shows that the major power saving comes from reduction on background power. When using our scheme, the background power dropped by 12.0%, 4.1% and 10.8% on average for MEM, ILP and MIX workloads, respectively. Figure 12 shows the percentage of time that any bank is open in a rank (rank active). If a bank is open, it may transfer data, wait for data bus ownership if another bank is transferring data or wait for DRAM timing constraints (tCL, tRTP, tWR etc) to expire to perform device operation. Our scheme targets the portion of a bank waiting for data bus and tries to minimize it. The result shows that the scheme can reduce the bank open rate by 13.7%, 9.9% and 12.2% on average for MEM, ILP and MIX workloads, respectively. For MEM workloads that have the most memory-intensive applications, the reduction on bank open rate ranges from 8.6% to 21.5%. Thus, the observation of reduction on background power is due to memory ranks staying longer at the precharge low-power state rather than at the active power state. For MEM workloads, the background power can be saved by 9.1% to 13.8%. For ILP workloads, the reduction on bank open rate is small (from 7.9% to

10.9%). This is because for CPU-intensive workloads, the parallel accesses to different memory banks are small. So the reduction on background power is also small (from 2.3% to 5.3%). The power consumption of MIX workloads is between that of the MEM and ILP workloads. The improvement of background power ranges from 6.5% to 14.0% because of the drop of bank open rate ranging from 6.1% to 21.6%.

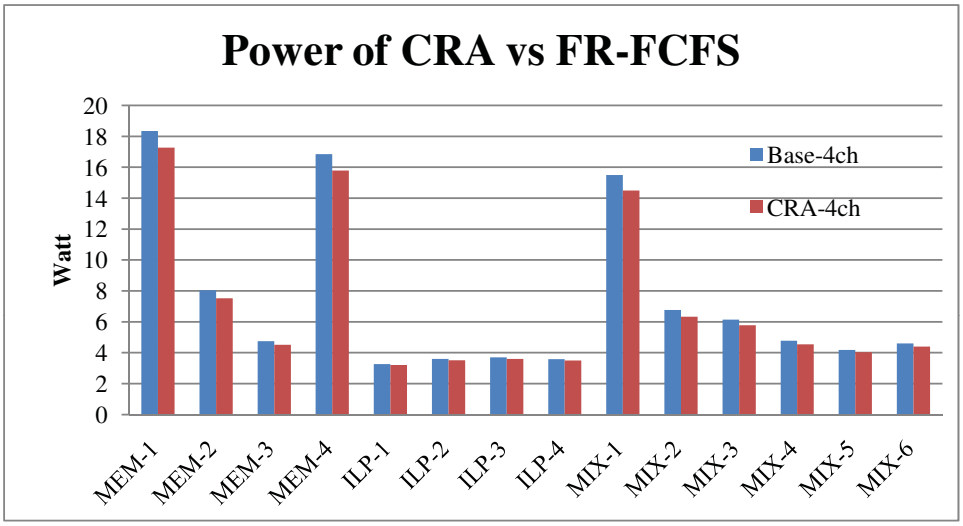
For operation power, I/O power and termination power, because our scheme does not change the underlying scheduler, their changes are small. The average difference on I/O and termination power for MEM, ILP and MIX workloads is less than 0.5%. Compared to the baseline, our scheme reduces the operation power by 3.3%, 1.7% and 1.7% for MEM, ILP and MIX workloads on average, respectively. This is due to the slight difference of power consumed by activating the first idle bank and the rest of the idle banks. Our scheme allows more banks to stay at the idle state and will increase the chance of activating a first idle bank in a rank.

3.4.4 Conservative Row Activation with More Memory Channels

We would like to see how the algorithm will perform in a system with more memory channels. Figure 13 is the result we get when the simulator is configured to have 4 Channels, 2 DIMMs per Channel and 2 Ranks per DIMM. The result is very similar to those in Section 3.4.1. The average power savings for MEM, ILP and MIX workloads are 5.9%, 2.4% and 5.3%, respectively. The average performance difference is less than 0.2% (our algorithm is better). We can see that the performance gain is smaller in the 4-channel configuration compared to the 2-channel configuration. This is because when there are more channels in the memory system, the contention on data bus will be less due to increased number of data buses. The higher utilization



(a) Weighted speedup of Conservative Row Activation compared to baseline for 4-channel configuration



(b) Power for Conservative Row Activation compared to baseline for 4-channel configuration

Figure 13: The power saving and performance impact of Conservative Row Activation under 4-channel configuration.

effect of our algorithm on data bus will be scattered and less distinctive. The power savings increased a little for MEM workloads and decreased for ILP and MIX workloads. Conservative Row Activation can only save memory power by letting the rank stay shorter in high-power active state, which is closely related to application's memory access pattern. Simply increasing the number of memory channels can not change the memory access patterns of applications much. So there is no significant change in power savings between 2-channel and 4-channel configurations. For memory-intensive workloads, the increase on power savings is mainly because larger channel bandwidth allows more requests being served by memory system and the chance for our algorithm working is higher. For other workloads, the channel bandwidth is not the bottleneck. In addition, more memory devices will increase the background power. Hence, the percentage value of power saving decreases.

3.5 Conclusion

In this study, we propose a new memory scheduling scheme called "Conservative Row Activation" that delays the row activation of a request for power optimization by allowing a bank to stay at the low-power mode until further delaying the activation would harm performance. It minimizes the performance impact by looking ahead when the bus becoming free and reserving slots on command/data buses for future column commands. Our simulation results indicate that the memory efficiency can be improved with negligible performance impact. This scheme can be implemented on top of many existing memory schedulers with little hardware overhead.

CHAPTER 4

HETEROGENEOUS MINI-RANK FOR OPTIMIZED PERFORMANCE AND POWER EFFICIENCY

4.1 Introduction

As the demands on memory capacity and bandwidth keep increasing, the memory power consumption has approached or even surpassed the processor power consumption in many server platforms. Thus, how to reduce the memory power consumption while still delivering high performance becomes an important issue for multi-core computer systems. A recent study proposed a mini-rank architecture that splits each DRAM rank into multiple mini-ranks and activates only one mini-rank for each memory request (96). Because fewer chips are involved for each request, the memory power consumption can be significantly reduced at the cost of slight increase on memory access latency.

A later study called Multicore DIMM (2) also divides devices in a conventional x64 rank into subsets, which they call "rank subsetting". It addresses the memory power efficiency issue using an approach similar to mini-rank but with a few differences. First, it splits the data bus among the rank subsets. Thus, the memory controller needs to be modified to support the change. In addition, if the accesses are not evenly distributed on subsets, the data bus of Multicore DIMM will be under-utilized. Mini-rank on the other hand, allows all the devices in a channel to have equal chance to use the data bus. Mini-rank uses a buffer to relay data

and commands between devices and bus; while Multicore DIMM uses a demux register to route control signals to devices and introduces less additional power and hardware cost.

Although the original mini-rank design works well for most workloads, we further observe that it has a limitation: it applies a fixed and unified partition from ranks to mini-ranks on all applications. Some applications will suffer a great performance loss under narrow mini-rank configurations, such as x8 mini-ranks; while others may lose power saving opportunities under wide mini-rank configurations, such as x32 mini-ranks. To address this issue, we further propose a heterogeneous mini-rank scheme that allows each application to have its own mini-rank configuration within a single memory system in order to approach the optimal power-performance trade-off. A latency-sensitive application will be assigned to a wide mini-rank configuration for maintaining its performance; while a latency-insensitive application will be assigned to a narrow mini-rank configuration for memory power saving. We find out that the memory bandwidth requirement of an application can be used as a guidance to select its near-optimal configuration.

For heterogeneous workloads containing applications with diverse memory access behaviors, on average, the heterogeneous mini-rank can reduce the memory power by 18.6% (up to 38.0%) with the performance loss of 2.4% (up to 7.4%), compared with a conventional memory system. In comparison, the x32 homogeneous mini-rank can only save memory power by 12.6% on average (up to 25.4%); while the x8 homogeneous mini-rank will cause the performance loss by 8.1% on average (up to 19.3%). Compared with the x16 homogeneous mini-rank configuration, the heterogeneous mini-rank can further reduce the EDP (energy-delay product) by up to 9.6%.

4.2 Heterogeneous Mini-Rank

The original mini-rank design applies a single configuration (e.g., x32 mini-ranks) to all workloads running in a single computer system. However, as applications may have different memory access behaviors from each other, their performance and power efficiency can be affected differently by the choice of mini-rank size. For instance, a workload of SPEC2000 applications of *wupwise*, *vpr*, *mcf* and *parser* sees 21.6% memory power reduction and 5.1% performance loss under the x32 mini-rank configuration, and 38.4% power saving and 26.9% performance loss under the x8 mini-rank configuration, respectively; while another workload of *swim*, *applu*, *art* and *lucas* sees 26.5% memory power reduction and 0.5% performance loss under the x32 configuration, and 52% power saving and 2.7% performance loss under the x8 configurations, respectively. It is obvious that different workloads may have different optimal mini-rank configurations, and thus assigning a fixed mini-rank configuration to the whole memory system may not exploit the full potential of mini-rank.

To address the limitation of the original mini-rank design (called homogeneous mini-rank in the rest of discussion), we further propose a dynamic heterogeneous mini-rank scheme that dynamically assigns different mini-rank configurations to different workloads. To implement this scheme, we have addressed the following design questions:

1. How to make online prediction of the best mini-rank type for each application?
2. How to map requests to its assigned mini-rank configuration?
3. How to implement in hardware and software to let multiple mini-rank configurations co-exist in a system?

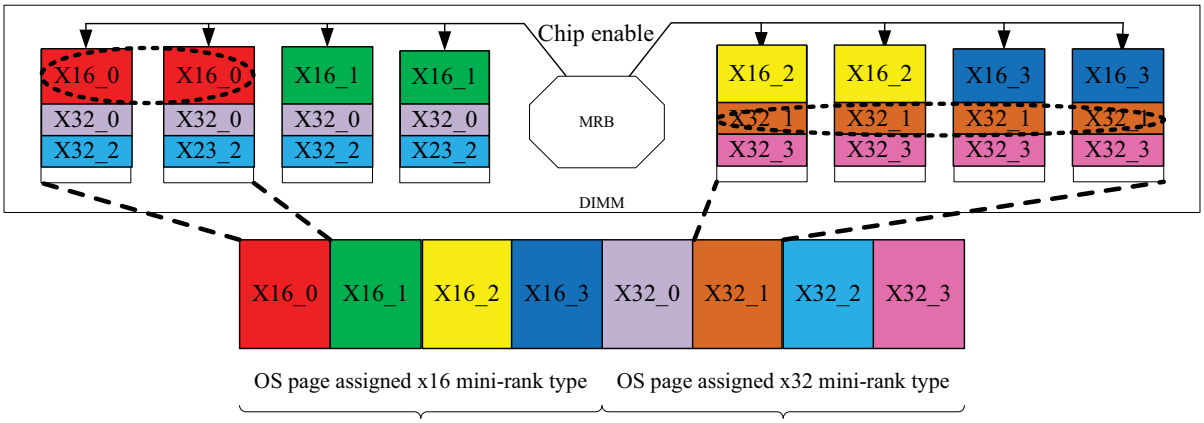


Figure 14: Data layout for heterogeneous mini-rank. Four x16 configuration and four x32 configuration sharing eight x8 device. The blocks of the same color represent all the data in a read/write burst. x16 configuration will burst using 2 devices for 16 memory cycles while x32 configuration will burst using 4 devices for 8 memory cycles.

4.2.0.1 Dynamic Mini-Rank Type Prediction

For the first design question, we have found that the memory bandwidth usage of a given application for the past time window is an effective predictor for the optimal mini-rank type for the application. The underlying reason is that applications of high memory bandwidth usage tend to have more concurrent memory accesses than others, and thus more requests can be served concurrently by using a narrow mini-rank configuration. Furthermore, they are not sensitive to the increase of memory access latency. By contrast, applications with moderate memory bandwidth usage generally have lower memory concurrency and are more sensitive to the increase of the latency. Thus, for memory-intensive applications of high bandwidth usage, using a narrower mini-rank can be more effective in power reduction and with lower performance loss.

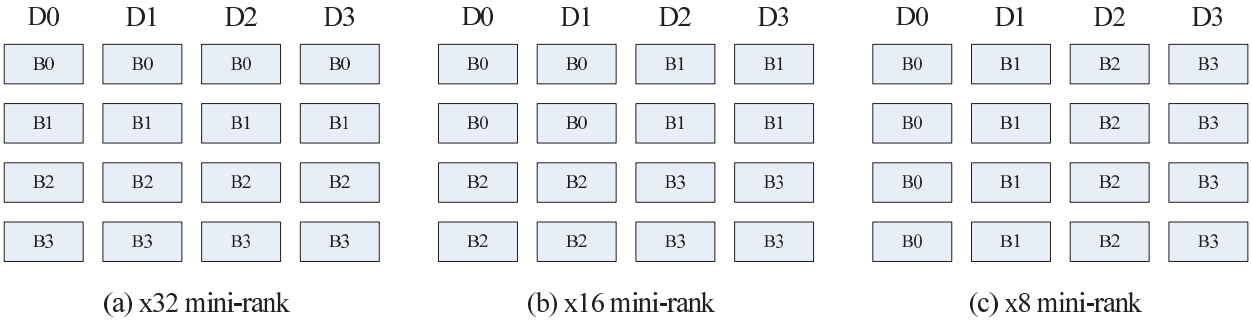


Figure 15: A view of memory block layout of heterogeneous mini-rank, assuming 64-byte cache line size and x8 memory device. D0, D1, D2, and D3 are four memory devices. B0, B1, B2 and B3 are four memory blocks of cache line size; and each square in the figure represents a sub-block of 16 bytes from those blocks. Those memory blocks may not necessarily be consecutive in the physical memory address space. A physical memory page of typical 8KB size may consist of 64 memory blocks, and their sub-blocks may be mapped onto a number of memory devices according to the memory interleaving scheme and the number of memory channels, DIMMs, and ranks in the system.

In the new design, the memory controller monitors the bandwidth usage of each application and records the information in a set of hardware counter registers. The operating system reads those registers and uses the information to predict the mini-rank type when a page is about to be loaded into physical memory. Physical memory pages are partitioned into three mini-rank types, namely x32, x16 and x8. The full rank size is not used because it is not power-efficient and its performance is very close to that of the x32 mini-rank size. We assume that on a page fault, the operating system allocates a page from the memory region of the predicted mini-rank type. All request addresses falling into that page will be accessed using this mini-rank configuration until the page is de-allocated. Section 4.2.0.2 discusses the memory block layout in details. In our simulation, the mini-rank type of a physical memory page is decided when a

virtual memory page of a given program is touched for the first time and then mapped to the physical page. In a real system, the operating system may use additional data structures to track pages of those three types.

4.2.0.2 Device Selection and Memory Layout

Contemporary memory systems distribute memory requests evenly across all channels, DIMMs, ranks and banks to maximize memory-level concurrency. For heterogeneous mini-rank, the same principle remains to be held. Consider a system with a last-level cache of 64-byte cache lines and a DDRx memory of x8 memory devices. With conventional memory, a cache miss will trigger a memory request to fetch 64-byte memory data, with 8 bytes from each device. With heterogeneous mini-rank, fewer devices are involved on a memory request, and the number of device varies with the mini-rank configuration. Figure 14 shows an example of high-level data layout of heterogeneous mini-rank with x8 devices. To simplify the example, we only show how the x16 and x32 configurations can be mixed together. Figure 15 further compares the layout of memory blocks from the same physical memory region under different mini-rank configurations, where the memory region size is a multiple of four times the memory page size. As it shows, the layout of a memory region can be changed without affecting the layout of other memory regions mapped onto the same set of devices.

Figure 16 presents the system view of a heterogeneous mini-rank system in our specific design. Two Type Mapping Tables (TMT) in Operating System (OSTMT) and in memory controller (MEMTMT) are used to keep tracking the mini-rank type for each physical memory page.

The OSTMT in our design is a table to record the mini-rank type of memory regions. The table is indexed with the physical memory address. Each entry in the table represents a memory region, which consists of multiple physical memory pages (4MB by default), and its mini-rank type. We use a large region size to reduce the overhead of OSTMT.

When the mini-rank type prediction is made at the page allocation, the OS finds an available slot in OSTMT and sets up its virtual address to physical address mapping accordingly. Note that all the pages belong to the same entry have the same mini-rank configuration but they may come from different applications. The management mechanism would be the same as how the OS manages super pages.

The MEMTMT is a mirror image of the OSTMT. It is modified by the OS when the OSTMT is updated, which only occurs when a page fault happens, as in our design the mini-rank type of a page stays the same until it is swapped out. When the memory controller receives a request, it looks up the MEMTMT using the physical address of the request to obtain the mini-rank type, and then determines the channel, DIMM and mini-rank that the request is mapped to according to the mini-rank type. The MRB will then generate chip enable signals to corresponding devices.

4.2.1 Memory Access Scheduling for Mini-Rank

The use of mini-rank affects two parameters in memory access scheduling, the interval between the read/write command and the data burst, and the number of data burst cycles. Figure 17 shows the difference of scheduling among a request of conventional x64 rank, a request of x32 mini-rank and a request of x16 mini-rank, assuming the accessed banks are activated.

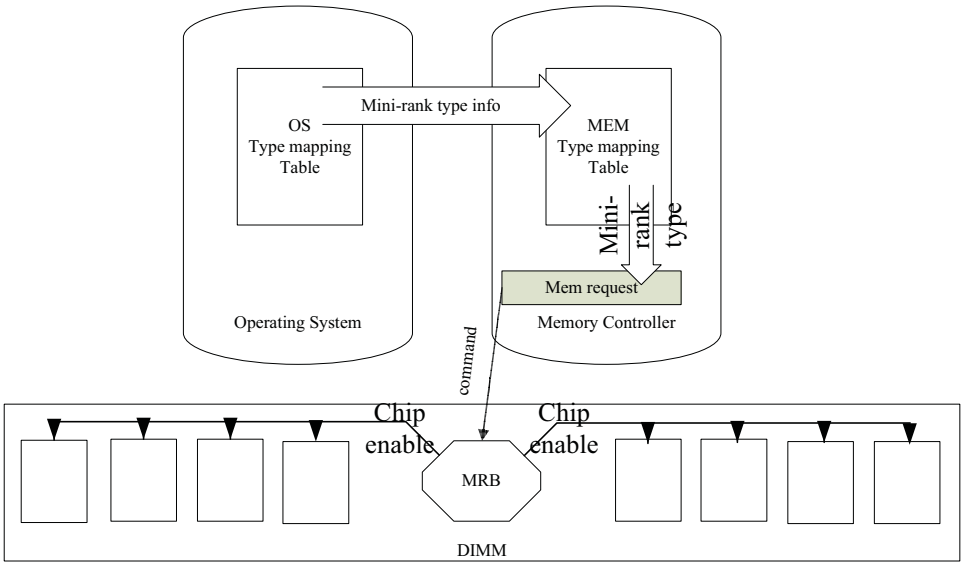


Figure 16: Basic structure of a heterogeneous mini-rank system.

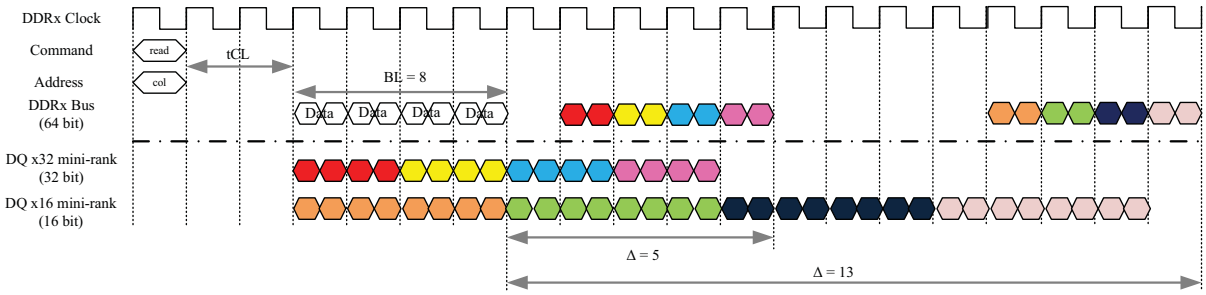


Figure 17: Timing of one request with the x32 mini-rank type and another request with the x16 mini-rank type. For conventional x64 rank, burst length (BL) is 8 so the data transfer take 4 cycles on DDRx bus. The x32 configuration will need additional 8 cycles to transfer data to MRB and 3 cycles of DDRx bus cycles can be overlapped. So the increased data transfer time is 5 cycles. For x16 mini-rank configuration, data will take 16 cycles to MRB and with 3 cycle overlap, the overhead is 13 cycles.

The scheduling for write accesses is similar. The timing parameter tCL is multiple cycles not shown fully in the figure. With conventional DDRx memories, the data burst takes four bus cycles. With x32 mini-rank, the data are firstly burst into the MRB for eight bus cycles, and then onto the x64 bus for four cycles in a pipelined manner. There is one extra cycle delay for the MRB to buffer the last trunk of data. Therefore, the latency is increased by five bus cycles over the conventional DDRx memories. The timing for x16 mini-rank is similar, except that the latency is increased by 13 bus cycles. There is another one-cycle latency for buffering the command/address signals. For heterogeneous mini-rank, it can schedule the timing for each request concurrently according to their mini-rank type information.

4.2.2 Mini-Rank Overhead

4.2.2.1 Mini-Rank Buffer Power Overhead

We first model MRB using Verilog and then break its power consumption into three portions: (1) I/O interface with DRAM chips and DDRx bus, including data, control and address pins; (2) DLL (Delay-Locked Loop); and (3) non-I/O logics, including SRAM data entries and re-decode logic. We estimated the I/O power by calculating the DC power using Thevenin equivalent circuit. The DLL power is derived from the power difference of DRAM device in two states with DLL enabled and disabled. The non-I/O logic power is estimated by the Synopsys Design Compiler (84) and standard 90nm technology library from UMC (1), which takes our Verilog code as input. It includes both static power and dynamic power and assumes average transistor activity level. Our simulator provides the activity ratio of the MRB chips, which is factored into the calculation of the actual I/O power and non-I/O logic power.

With memory sub-system configuration of four channels, two DIMMs per channel, two ranks per DIMM and two mini-ranks per rank (x8 devices) with 50% read and 30% write channel bandwidth utilization (emulating memory intensive workloads), the I/O power, non-I/O logic power, and DLL power contributes to 85.9%, 8.7% and 5.4% of the MRB power, respectively; and the total MRB power is 936mW. Not surprisingly, the I/O power dominates. Nevertheless, the I/O power between the MRB and the DDRx bus, which is 58.1% of whole MRB power, is also needed in conventional DDRx systems. The actual power increase is 394mW per DIMM for the configuration.

4.2.2.2 Overhead for Managing Type Mapping Tables

The OSTMT and MEMTMT may be updated only upon a page fault. When a page fault happens, the OS will perform a sequence of jobs, including updating the page table and swapping pages between main memory and disks. Managing the OSTMT only requires the OS to consider the mini-rank type of an application and assign it to the corresponding page. If an entry of the same type is not full yet, the page will be assigned to the entry and its physical address will be determined. In this situation, no information is needed to be sent to the MEMTMT. If all the entries of the same type are full, a new entry will be allocated either by selecting an empty one or replacing an existing one (all the pages in the replaced entry will be swapped out and the policy can be adopted from how OS manages super pages). After that, a port write operation will be used to update the corresponding entry in the MEMTMT. The OSTMT can either be integrated into the page table (two additional bits to indicate the mini-rank type) or

stand-alone. Compared to all the other operations that a page fault handler needs to perform, the time overhead of managing the OSTMT and MEMTMT is negligible.

4.2.2.3 Hardware Overhead of MEMTMT

The MEMTMT in the memory controller contains the mapping from physical addresses to mini-rank types. As the most significant bits of physical addresses are used as the index to the MEMTMT, no space is needed to store the address information; and only three bits per entry are required. A major design question is to decide how large the address region that an entry should represent. The smaller region the entry represents, the finer grain the management can have, but the larger the MEMTMT size. Take a system with 8 GB of physical memory as an example: if each entry represents 4MB of memory space, the MEMTMT will have 2K entries and occupy 6K bits.

4.3 Experimental Methodologies

We built a detailed DDR3 DRAM memory model and integrated it into MARSSx86 (63) simulator. The simulator is capable of keeping track of each request and the state of every memory channel, rank and bank. The simulated memory controller will issue commands according to the current memory status and pending requests. We implemented a read-first and hit-first scheduling policy that groups all bank hits and issue them together on top of FR-FCFS scheduler. The scheduler will issue the column access command together with an auto-precharge if the column command comes form the last row-hit request of a bank. This greedy policy maximize bank hit and precharge a bank to minimize miss latency of that bank. The memory transactions are pipelined whenever possible. Also the XOR-based mapping (95) is used to

TABLE IV: Major simulation parameters for Heterogeneous Mini-rank.

Parameter	Value
Processor	4core, 3.2GHz, 4-issue per core, 14-stage pipeline
Functional units	2 IntALU, 4 LSU, 2 FPALU,
IQ, ROB and LSQ size	IQ 32, ROB 128, LQ 48, SQ 44
Physical register num	128 Int, 128 FP, 128 BR, 128 ST
Branch prediction	Combined, 6k bimodal + 6k two level, 1K RAS, 4k-entry and 4-way BTB
L1 caches (per core)	64KB Inst/64KB Data, 2-way, 16B line, hit latency: 3-cycle Inst/3-cycle Data
L2 cache (shared)	4MB, 8-way, 64B line, 13-cycle hit latency
Memory	2 channels, 2 DIMMs/channel, 2 ranks/DIMM, 8 banks/rank
Memory controller	64-entry buffer, 15ns overhead
DDR3 DRAM latency	DDR3-1600:11-11-11: precharge/row access/column access: 13.75ns

maximize memory level parallelism. More specifically, in a system of 2 channels, 2 ranks per channel and 8 banks per rank, we select address bits 20, 22, 23, 24, 25 and XOR them with address bits 6, 7, 8, 9, 10. Then the lower bits 0-5 is the byte address, bit 6 selects the channel, bit 7 selects the rank and bits 8-10 select the bank; the upper bits are used as column index and row index. We also built a module to simulate the function of TMTs. Table IV shows the major simulation parameters.

TABLE V: Parameters used for calculating DRAM Power on Heterogeneous Mini-rank.

Parameters	Values
Normal voltage	1.5V
Active precharge current	95mA
Precharge power-down standby current	35mA
Precharge standby current	42mA
Active power-down standby current	40mA
Active standby current	45mA
Read burst current	180mA
Write burst current	185mA
Burst refresh current	215mA

4.3.1 Memory Power Calculation and Performance Metrics

We follow the Micron power calculation methodology (52) to estimate the power consumption of DDR3 DRAM devices. At the end of every memory cycle, the energy consumption of the current cycle is calculated based on the state of each device, and the result is recorded and accumulated. The parameters used for calculating the DRAM power are listed in Table V. For cases that the electrical current values presented in data-sheet are from the maximum device voltage, they are de-rated by the normal voltage (52). The MRB power is calculated as described in Section 4.2.2.1.

The performance is characterized using weighted speedup (80) $\sum_{i=1}^{n}(IPC_{multi}[i]/IPC_{single}[i])$, where n is the total number of instance running, IPC_{multi}[i] is the IPC value of the application running with other instances and IPC_{single}[i] is the IPC value of the same application running alone.

4.3.2 Workloads

Our experiments simulate a quad-core x86-64 system with Linux 2.6.31.4 (64 bit) running on it. The workloads is built the same as described in Section 3.3.2

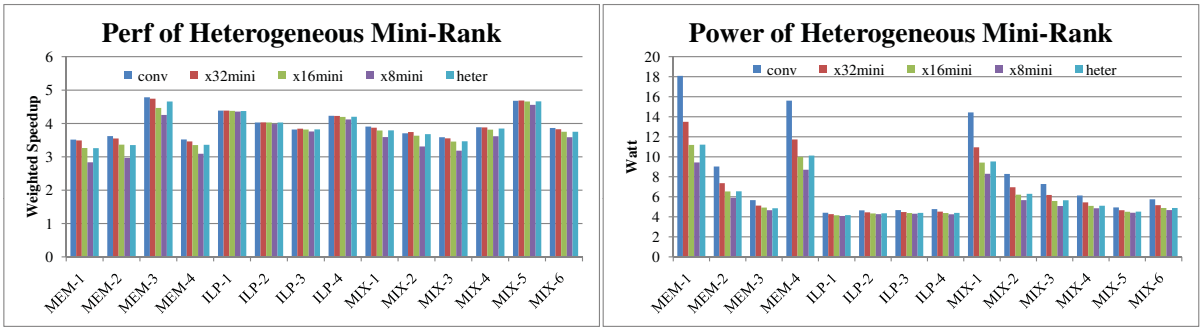
As discussed in section 4.2.0.1, the real time bandwidth usage of each application is used to determine its mini-rank configuration under heterogeneous mini-ranks. In our experiments, an application that consumes higher than 5GB/s memory bandwidth will use the x8 mini-rank configuration for newly allocated pages; an application whose bandwidth usage is between 3GB/s and 5GB/s will use the x16 configuration; and an application whose bandwidth usage is below 3GB/s will use the x32 configuration.

We inserted signal at the starting point of every application's main execution phase and let the application running in the fast forwarding mode until the signal is reached and then the application will sleep. When all the applications in a workload reach the signal, all the applications will be awakened and the detailed simulation and statistic collecting will launch for the following 400 million instructions.

4.4 Experimental Results

4.4.1 Evaluation of Heterogeneous Mini-Rank

Overall, heterogeneous mini-rank achieves a better trade-off between power efficiency and performance than homogeneous mini-rank. Figure 18 compares the performance and memory power consumption of the conventional x64 rank, homogeneous mini-rank configurations (x32, x16, x8) and the heterogeneous mini-rank configuration determined by workload's memory bandwidth requirement. Compared with the conventional x64 rank, heterogeneous mini-rank



(a) Weighted speedup for conventional, homogeneous and heterogeneous mini-rank configurations

(b) Power for conventional, homogeneous and heterogeneous mini-rank configurations

Figure 18: Overview of power saving and performance impact with varying mini-rank configurations and heterogeneous mini-rank configuration.

reduces memory power consumption by 5.4% to 38.0%, with a performance loss of -0.1% to 7.4%. In comparison, the x32 homogeneous mini-rank configuration reduces memory power consumption by 2.0% to 25.4%. The x8 homogeneous mini-rank reduces memory power consumption by 7.2% to 47.8%; however, it incurs performance loss by up to 19.3%, The x16 homogeneous mini-rank, in general, achieves better trade-off between power efficiency and performance than x8 and x32 ones; however, it does show weakness on some cases. For instance, for the MEM-3 workload, it reduces memory power consumption by 12.9% but at the cost of 6.7% performance loss, while the x32 configuration reduces power by 9.6% with 0.9% performance loss. In comparison, the heterogeneous mini-rank reduces memory power consumption by 14.3% with 2.6% performance loss.

Figure 19 shows the normalized EDP (Energy-Delay Product) for conventional, homogeneous and heterogeneous mini-rank configurations. Note that for EDP, the smaller the value the better. The heterogeneous mini-rank achieves almost the same or even better EDP compared to the best EDP values of homogeneous mini-rank. For workloads MEM-3, MIX-2 and MIX-4, the heterogeneous scheme has an EDP value that is 1.7%, 1.2% and 1.4% better than the best EDP value of homogeneous mini-rank, respectively. For workloads ILP-1 and MIX-1, the heterogeneous scheme is 1.2% and 3.0% worse than the best EDP of homogeneous configuration. The reason is that for the two workloads, homogeneous mini-rank produces the best EDP on x8 configuration. As described in Section 4.3.2, for all ILP workloads (memory bandwidth $<$ 500MB/s), we apply x32 mini-rank configuration. Hence the dynamic scheme cannot put the ILP applications in the two workloads to x8 configuration and miss the opportunity for the best EDP. However, adding x8 configuration will make the other ILP application perform much worse. Although the x8 homogeneous mini-rank gets 1.2% and 3.0% better EDP, the heterogeneous scheme is very close. As for the rest workloads, the EDP differences of best homogeneous mini-rank and the dynamic scheme is less than 0.1%. Note that for different workloads, the homogeneous mini-rank configuration with best EDP value might be different.

In summary, heterogeneous mini-rank can maintain the performance and reduce the memory power consumption for applications of a large spectrum. Homogeneous mini-rank achieves significant overall improvement of power efficiency; however, for individual workloads of diverse memory access characteristics, it may incur visible performance losses with small mini-rank size or under-utilize power saving opportunities with relatively large mini-rank size.

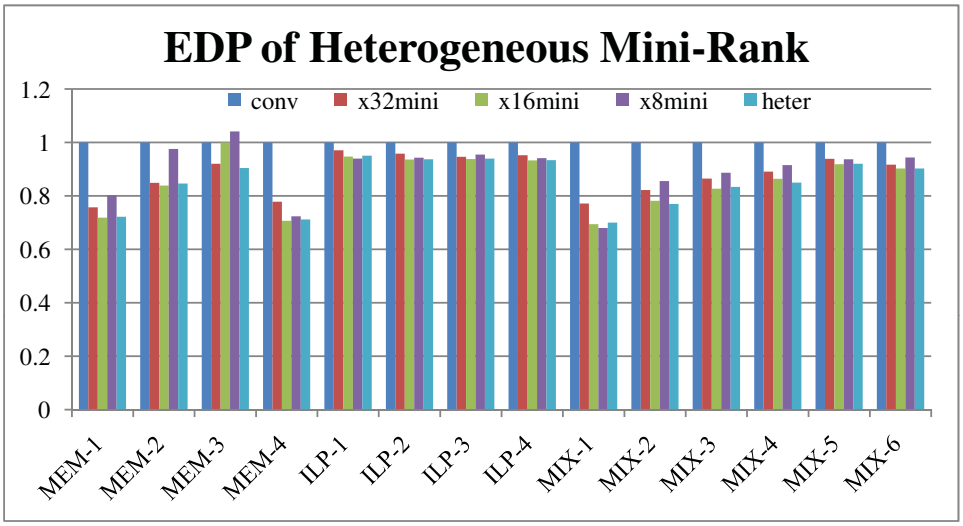


Figure 19: Normalized EDP (Energy-Delay Product) for conventional, homogeneous and heterogeneous mini-rank configurations. (Smaller is better)

4.4.2 Analysis of Heterogeneous Mini-Rank Performance Impact

In this section, we analyze the source of the performance gain of heterogeneous mini-rank compared with homogeneous ones. Figure 20 shows the read latency in memory cycles of each application under homogeneous and heterogeneous mini-ranks. We only present the results of workload MIX-1 here (including applications *lbm*, *libquantum*, *povray*, and *calculix*). The workload is selected because it has a diverse per application memory bandwidth (6.0GB/s, 4.8GB/s, 550MB/s and 102MB/s respectively). Thus, its heterogeneous configuration performs differently from the homogeneous configurations. Other workloads have similar results.

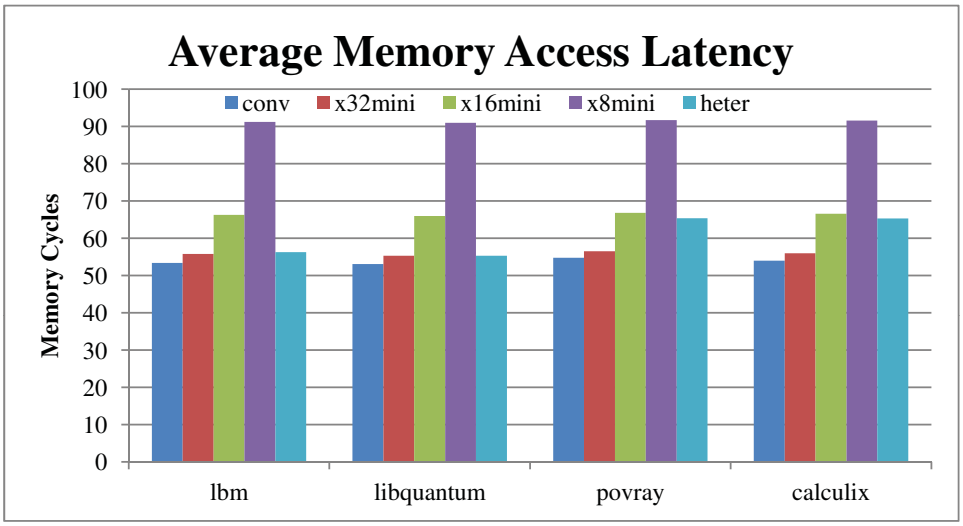


Figure 20: Read latency of individual application in workload MIX-1.

The result shows that under the homogeneous configurations, each application sees almost the same average read latency. While under the heterogeneous configuration, the MEM applications (*lbm* and *libquantum*) see shorter average read latency than the ILP applications (*povray* and *calculix*). Because the performance of ILP applications is not sensitive to the read latency, the heterogeneous mini-rank can better balance between performance and power consumption.

4.5 Conclusion

In this study, we propose a heterogeneous mini-rank scheme that can provide near optimal performance/power trade-off, and avoid big performance loss for workloads with diverse memory access behavior. The heterogeneous mini-rank configuration selection is based on application's run-time memory bandwidth usage. It is simple and only introduces small overhead in both software and hardware implementations.

CHAPTER 5

MEMORY ARCHITECTURE FOR INTEGRATING EMERGING MEMORY TECHNOLOGIES

5.1 Introduction

Main memory system design is on the edge of revolutionary changes. On one hand, DRAM-based memory systems are stretched to meet the increasing demands on high memory bandwidth and large memory capacity from multi-core processors. As a negative side-effect, memory power consumption and overheating have become design constraints for many server platforms. On the other hand, emerging memory technologies present new opportunities to address those issues. For instance, Phase-Change Memory (PCM), Spin Torque Transfer Magnetic RAM (STT-MRAM), Nanowire Phase Change Memory (NW/PCM), Fuse/Antifuse memory technology are promising alternatives to the DRAM technology.

Those new developments have pushed the existing memory system design to its limit. DDRx memory systems, which are dominant in laptops, personal computers, workstations and servers, evolve from SDRAM (synchronous DRAM) of decades ago. While many optimizations have been integrated into DDRx and memory bandwidth has been improved by more than twenty-fold, the memory organization is almost unchanged. The memory controller has to maintain rigid control on all memory devices, schedule almost all device-level operations including precharge, activation (row access), data read/write (column access) to meet device timing con-

straints and avoid bus contentions. To do so, the controller has to track the status of internal banks of all devices, such as row buffer contents, power modes and progress of ongoing operations.

This rigid memory access protocol presents a severe problem to integrating new memory technologies into existing memory systems. Those technologies behave very differently from DRAM; and many of them need complicated internal structures to work efficiently. Consider the PCM design as an example: all proposed designs in the research domain use some kinds of buffers, either inside the devices or the memory modules, to alleviate the limited write endurance and long latency of PCM devices (41; 99; 69; 67). In other words, those memory modules are advanced modules with complex internal structures. To apply a rigid memory access protocol as before, the memory controller will have to track the status of both PCM storage cores and their buffers, plus the progress of all ongoing operations. Furthermore, the memory controller must store the address tags of those buffers; otherwise, the controller cannot know whether a request hits in a buffer and thus its access latency, which is required to schedule the bus transactions.

Recent research in memory systems shows a trend of using decoupled memory organization, in which a bridge chip is added to a memory module to relay command/address and data between the memory controller and DDRx DRAM devices for improving scalability, performance or power efficiency. For instance, Fully-Buffed DIMM (FB-DIMM) uses a narrow, high-speed memory channel organized as a daisy chain with a bridge chip to convert between the FB-DIMM channel and on-DIMM DDRx bus (86). Registered DIMM (54) uses a buffer on each DIMM to

reduce the electrical load on the command/address bus so that more DIMMs can be installed on a memory channel. MetaRAM (51) uses a controller to relay address/command and data between the controller and devices, so as to reduce the number of externally visible ranks on a DIMM and reduce the load on the DDRx bus. Mini-Rank (96) and Multicore DIMM (2) activate sub-ranks narrower than the 64-bit bus to save memory power. The bridge chip is used to relay data between the DDRx bus and sub-ranks. Decoupled DIMM (97) uses memory devices with low data rate and bus with high data rate to improve performance. The bridge chip is used to convert between the two data rates.

However, decoupled memory organizations do not enable universal interoperability for diverse memory modules. Their bridge chips simply relay command/address and/or data, with memory controllers scheduling device operations as before. Without universal interoperability, future systems may have to use different memory controllers for various types of memory modules, which is infeasible in economic sense. Furthermore, future processors are likely to use integrated memory controllers, which means there would be many sub-types of processors.

We propose *Universal Memory Architecture* (UniMA) to enable *universal interoperability* between all major processors and memory modules made by emerging memory technologies as well as DRAM.

The UniMA proposed in this study is a *framework* of memory architecture rather than a particular implementation. It is an advanced memory organization with a new memory access protocol that offloads the scheduling of device-level operations to each memory module. As for the implementation, it extends the logic functions of the bridge chip when compared with

the decoupled memory organization. Such an extension does not incur significant extra cost or power consumption, and it inherits the improved scalability, performance and power efficiency from decoupled memory organizations.

In this study, we evaluate an implementation of the UniMA framework on top of the DDRx bus protocol, so that we can compare it with existing memory system designs. Each memory module embeds a bridge chip that performs local management, e.g., for DRAM devices, to schedule device operations including precharge, activation and read/write. The memory controller may still perform memory access reordering, i.e. reorder memory requests based on some scheduling schemes, but without timing constraints. The bridge chip is an extra chip on the memory module like those on the FB-DIMM, Mini-Rank and Decoupled DIMM, or the bottom chip in future stacked 3-D memory modules. Since the controller no longer has the full knowledge of each device's status, a new protocol is proposed for the communication between the controller and modules to avoid bus contentions. The controller sends generic commands such as read or write instead of device-specific ones to modules; a module with ready data will raise its readiness via some additional signal lines; and a token-based approach is used to grant bus ownership to one ready module and avoid contentions.

Our simulation results show that UniMA achieves comparable performance when compared with conventional DDR3 memory systems. On a simulated quad-core system with homogeneous DRAM devices, UniMA can improve performance by 3.1% on average (up to 4.5%) for memory-intensive workloads; and incurs an average performance loss of 1.0% (up to 1.8%) for none memory-intensive workloads.

UniMA may also support heterogeneous devices in one system, which is not feasible under the conventional memory organization. A heterogeneous system of PCM and DRAM, for example, may balance the large capacity and relatively high power efficiency of PCM with the relatively high performance of DRAM. Our simulation results show that compared with the performance of a pure DRAM system, the overall performance of a pure PCM system is about 25% lower for memory-intensive workloads, while the performance of a hybrid system enabled by our UniMA design is only 12% lower.

The main focus of this study is to answer the question whether the UniMA framework featured with localized scheduling is a viable approach. The UniMA implementation here is for evaluation purpose only. It is not fully optimized and is not intended to be a standard. Particularly, it does not introduce any major change of memory bus design, though a new bus design may be expected with UniMA. Neither does it evaluate the optimal data layout for hybrid memory systems, which may further improve the performance of such systems. Both issues are very complicated and beyond the scope of this study.

5.2 Prototype Design of UniMA

In this section, we present a prototype design of UniMA on the conventional DDRx memory channel for evaluation purpose. UniMA is a memory architecture framework, which can also be applied to other memory channel designs, e.g. FB-DIMM or Rambus channel. We give this design to demonstrate that UniMA is implementable; and our evaluation results will show that it can achieve efficiency comparable to that of the conventional memory organization. We

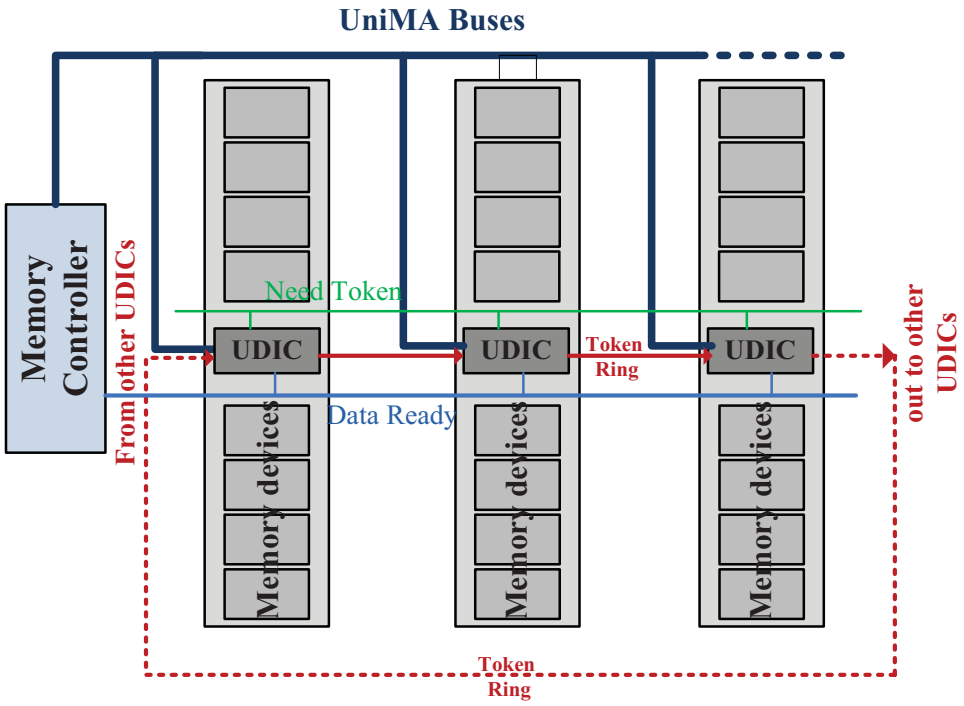


Figure 21: The organization of UniMA design with single channel configuration.

minimize the changes on DDRx memory channel to make a fair comparison. If UniMA is to be employed in product systems, the actual implementation can be very different.

5.2.1 Prototype Design Overview

Figure 21 shows the organization of UniMA applied to a DDRx memory system. We have several design considerations. First, timing and other device-specific management issues should be transparent to the memory controller for interoperability. Since the controller no longer tracks the status of each DIMM and ongoing operations, a new protocol is needed to avoid contention on the shared data bus and to notify the memory controller when fetched data will

be ready. We propose a token-based approach to grant data bus ownership to ready DIMMs and avoid data bus contentions. The details will be discussed in Section 5.2.2.2.

We have also considered other design alternatives on this aspect. One approach is to add a bus arbitration chip onto the motherboard to handle data bus contentions. However, this would require modifications to the controller, DIMMs and a dedicated chip on board, and is not cost-effective and efficient. Another approach is to use time-division multiplexing to avoid bus contentions. The controller may assign a fixed time slot to each DIMM for reporting whether it has ready data and then schedule ready transfers accordingly. However, this approach is not scalable. Another design consideration is to minimize the changes to the current DDRx bus design for the reasons discussed above.

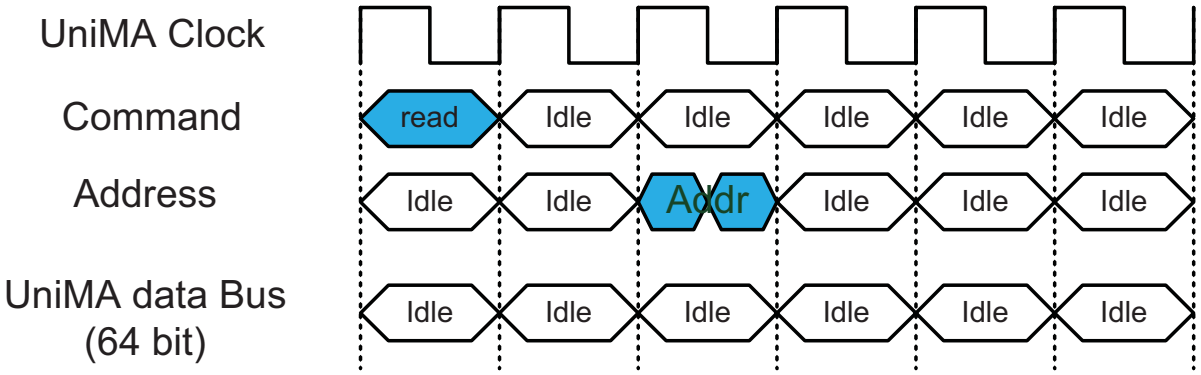
A new memory access protocol, including a set of device-independent commands, is defined to support the communication between the memory controller and DIMMs. A bridge chip called Unified DIMM Interface Chip (UDIC) is added to each DIMM to process the communication. Three additional lines are added to avoid contentions on the shared data bus. All DIMMs connected to a channel form a ring by a "Token Ring" line that passes a token between two adjacent DIMMs. Only the DIMM holding the token can use the data bus. A "Data Ready" line is used to notify the controller that a DIMM has fetched data ready. It is shared by all DIMMs in a channel; and an "OR" operation is performed on the line. Finally, a "Need Token" line is added to avoid unnecessary token passing operations, which is only set when a DIMM without the token has data ready. The DIMM holding the token will only pass the token to others when the line is set.

5.2.2 Memory Access Protocol

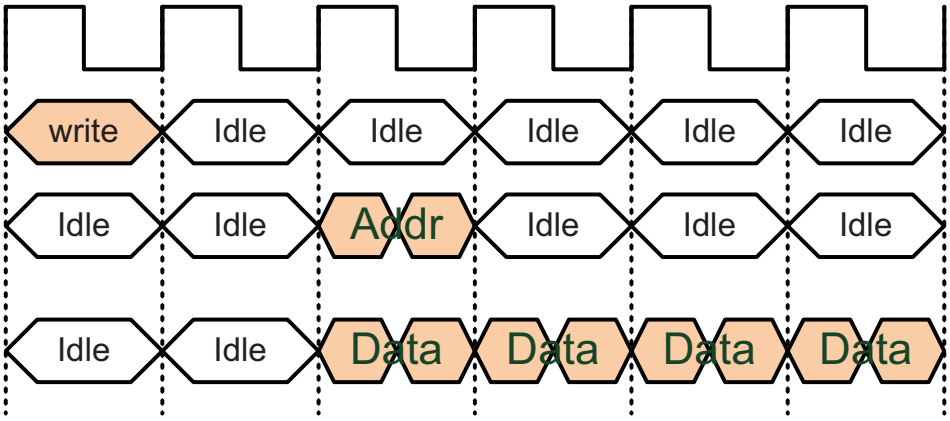
5.2.2.1 Memory Commands

We will first discuss the design of device-independent commands to support communication between the controller and memory modules, and then discuss how to use a token-based scheme to avoid data bus contention. The core command set generated by the controller only includes "Read", "Write", and "Get Read" commands. An extended set can be used for configuration. Figure 22 shows an example of timing regarding those commands. A delay of one memory cycle is inserted between a command and its address/data transfer to allow the bridge chip (UDIC) to set bus direction accordingly after receiving the command. When the memory controller receives a read or write request from the processor, it will send out a "Read" or "Write" command with the address (and data for a write request). The UDIC on the destination module will buffer the command and address (with data for a write), and generate corresponding device-level commands when the devices are ready.

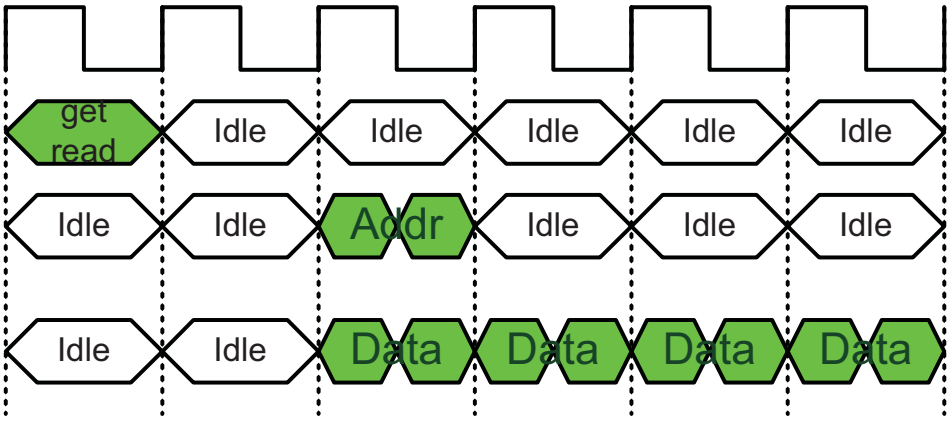
For a write request, after the controller sends out the command, address and data to the corresponding module, their communication for this request is completed. For a read request, additional communication is required for sending the fetched data back. In a conventional memory system, the controller will only issue the read command to a DIMM if the bus will be free at the time of data returning, in addition to other timing constraints. This is no longer the case in UniMA. The issue of "Read" command from the controller and the read operation on devices are decoupled. When a read request is finished at a module, the module needs to notify the controller that it has data ready to be sent back. To do so, a "Data Ready" line is added



(a) UniMA Read command



(b) UniMA Write command



(c) UniMA Get Read command

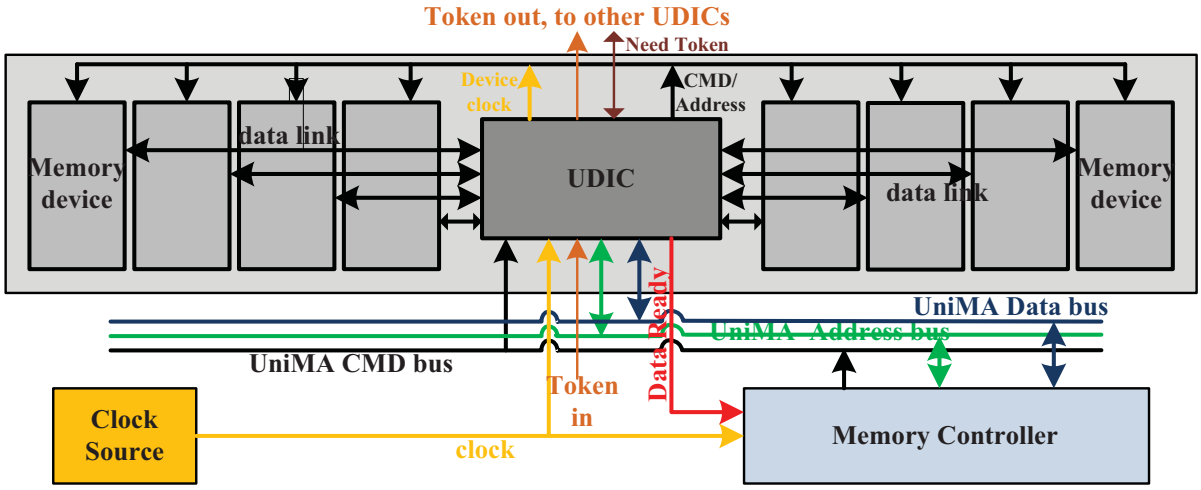
Figure 22: The timing of commands (Read, Write and Get Read) for UniMA design.

and is shared by all the DIMMs on a channel. A UDIC sets the line when it has or will have data ready for transfer. Note that since the UDIC schedules local device operations, it knows in advance when the data will be ready. It may set the line in advance as long as the data will be ready when the bus becomes free[1]. The controller monitors the "Data Ready" line and issues a "Get Read" command when it can accept returning data. There might be multiple DIMMs with data ready; only the one holding the token can respond to the "Get Read" command and send its ready data back to the controller.

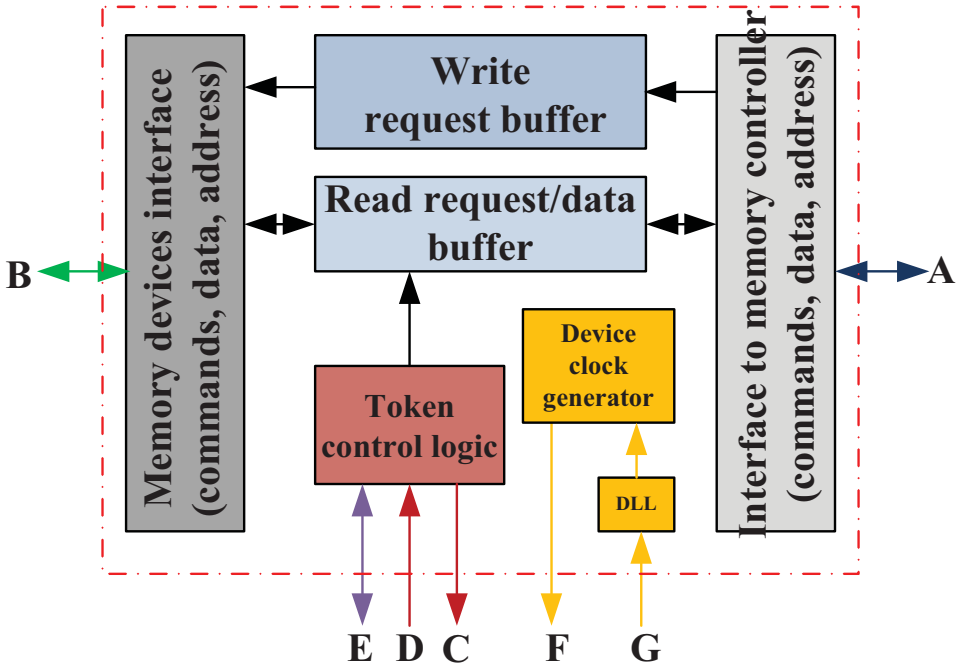
Since the controller does not know which read request is returned, the DIMM also needs to send that information back with the data. One design choice is to add a request ID to each pending request. This would require a few additional lines to transfer the request ID between the controller and DIMMs. Another solution is to send back the address with each fetched data. This will increase the traffic on the address bus. In a typical DDRx system with the L2 cache block size of 64 Bytes, each read request will take two memory cycles to transfer the address (row and column) and four memory cycles to transfer the data. Under this design, each read request needs to transfer its address twice and would cause new bottleneck on the address bus. This is because now the address bus is bi-directional and one cycle of high impedance on pins is required when the direction of signal transfer changes (73). To avoid the bottleneck, the same double pumping mechanism as that used on the data bus can be applied on the address bus, so that each address transfer only takes one memory cycle.

---

[1]We will discuss later how the UDIC knows when the bus becomes free.



(a) UniMA DIMM design and connections.



(b) UniMA UDIC design.

Figure 23: Detailed design of UniMA DIMM and UDIC. **A**: connecting to memory controller; **B**: connecting to devices on DIMM; **C**: connecting to the next UDIC on the "Token Ring"; **D**: connecting to the previous UDIC on the "Token Ring"; **E**: setting or receiving "Need Token" signal; **F**: clock for devices on DIMM; **G**: incoming clock from bus.

5.2.2.2 Token Ring for Coordinated Data Bus Scheduling

In order to resolve the contention on data bus, a single token is assigned to each channel, which is a one-bit flag indicating the ownership of data bus. A ring is formed to pass the token between adjacent DIMMs as discussed above. When the controller issues a "Get Read" command, only the UDIC holding the token can burst its data (normally the oldest ready one) to the shared data bus. To maintain fairness, it will also check the "Need Token" line and pass the token to the next UDIC if that line is set. The line can be set by any other DIMM that needs the token to transfer data. If the line is not set, the current owner will hold the token to avoid unnecessary token passing.

To minimize the token passing overhead and maximize the data bus utilization, when a UDIC receives the token, it can grab the token if it already has data buffered or will have data ready before the data bus becomes free; otherwise, it will pass the token to the next UDIC. As mentioned above, a UDIC knows in advance when its read data becomes ready; and it uses an internal timer to monitor when the data bus will be free. For any UDIC without the token, every time when it observes a "Get Read" or "Write" command on the command bus, it will set its timer to four for a system with last-level cache block size of 64 Bytes and decrease the timer by one on every memory cycle until that reaches zero. The timer indicates when the data bus becomes free since every data burst takes four memory cycles.

5.2.3 Unified DIMM Interface Chip

The Unified DIMM Interface Chip (UDIC) acts as the bridge between the controller and memory devices. It receives device-independent operation commands from the controller, sched-

ules and issues device-level commands accordingly, and sends read data back to the controller. Figure 23b shows the major components of the UDIC. The *Interface to Memory Controller* is responsible for receiving commands and write data from the controller and sending read data back. A *Read Buffer* holds the addresses received from the controller and the data returned from the devices for read requests. A *Write Buffer* stores the addresses and data of write requests received from the controller. Each buffer has 32 entries in our experiments. The *Device Interface* is the functional module of UDIC to control memory devices on DIMM. It controls the timing of devices and generates device-level commands accordingly. The *Token Control Logic* is responsible to request and pass the token. Finally, a *DLL (Delay-Locked Loop) Logic* is used to reduce the clock skew from the memory bus; and a *Device Clock Generator* generates the clock signal for the devices since they may run at a rate different from the bus clock rate.

5.2.4 Overheads

The UDIC on each DIMM will introduce some cost, but its complexity is much lower than that of a conventional memory controller. A conventional memory controller must handle the maximum number of DIMMs, ranks, and banks that the processor supports, while a UDIC only handles a single DIMM. Also, conventional memory controller supports both device timing and request reordering for all requests, but the UDIC only maintains the timing constraints for local devices. The complexity of memory controller can also be reduced because the device timing function is moved to the UDIC. The controller may use a small counter for each UDIC to track its fullness state to avoid overflowing UDICs. Moving device timing to the UDIC is the key that enables our UniMA design to support diverse memory technologies under a universal

architecture. A different approach to supporting diverse memory modules in one system is to use a dedicated memory controller for each type of memory modules. However, this approach is not efficient and hard to be implemented.

Buffering requests and data on the UDIC will increase the idle latency of a single memory request. However, because memory requests can be served in parallel and pipelined, the overall performance overhead is small. In terms of additional power consumption, sending the address back with the read data consumes some extra power. However, considering that the address bits are much narrower than a typical cache block, the extra power is small in percentage. The UDIC also consumes additional power. We use the methodology in existing studies (96; 97) to estimate its power consumption. The power consumed by the I/O interface to the controller and devices is estimated by using Thevenin equivalent circuit. The non-I/O logic power (including static and dynamic power) is generated by Synopsys Design Compiler (84) using our Verilog files as input. Using the configuration of 533MHz bus speed and standard 90nm technology library from UMC (1), the I/O power for bus transfer is the dominant, with 804mW when the memory traffic is high (assuming 50% read and 30% write channel bandwidth utilization). The overall power consumption for UDIC is 1.66W. Considering that the conventional DDRx memory system also needs to consume the same I/O power for the bus transfer, the additional power consumption of UDIC with bus overhead is only 856mW. Note that the additional power will drop with the decrease of memory traffic.

5.2.5 Other Discussions

The device voltage and clock frequency of different memory technologies may vary. In order to support diverse memory devices, different voltage levels and clock rates may be needed. The UDIC can easily generate multiple clock rates using the clock generator and the bus clock, but not multiple voltage levels. A possible design choice is to let the DIMM socket on motherboard provide multiple voltage levels. This may increase the pin count of DIMM. Another alternative is to provide the standard voltage only but put a voltage regulator on each DIMM to generate the suitable power source for devices. This would increase the DIMM implementation cost and generate additional power and heat. The regulator can be either stand-alone or integrated with the UDIC. In the later case, off-chip inductors and (usually) off-chip capacitors are needed (64). The stand-alone regulator generally has an efficiency of 70% to 80% (76), while the integrated one may achieve a higher efficiency of around 80% to 90% (64). The power consumption of the regulator can be calculated as $((1 - E_r)/E_r) * P_d$, while E_r is the efficiency of the voltage regulator and P_d is the total power consumption of devices on DIMM.

5.3 Experimental Methodologies

We built a detailed memory simulator for the conventional DDRx and our proposed UniMA design, and integrated it into M5 (6) simulator. For experiments of conventional memory systems, the simulated memory controller will issue device-level commands based on the status of pending requests and memory channels, ranks and banks, using the hit-first and read-first scheduling policy. For experiments of UniMA design, the memory controller only monitors the bus state and issues UniMA commands to maximize the bus utilization. It also groups

requests with close addresses to increase the possibility of row buffer hits and allows reads to bypass writes. The simulated UDIC on each DIMM is then responsible to translate the UniMA commands into standard device-level commands. In our experiment, memory transactions are pipelined whenever possible. For comparison, we also simulate a simplified PCM-based system, and call it pseudo-PCM. For the PCM DIMM design and timing, we use the same assumption as that in a previous study (41): the PCM DIMM has similar device architecture as DDRx but has different cell organization. The timing for PCM device is adopted from existing studies (41; 66). A PCM device is only precharged when its row buffer is dirty and a new row request is pending due to its non-volatile feature. Table VI shows the major simulation parameters.

In our experiments, we simulate a quad-core system with each core running a distinct application from SPEC2000 and SPEC2006 benchmarks (81). We follow the method used in an existing study (38), to group those benchmarks into two categories: MEM (memory-intensive) and ILP (compute-intensive) based on their numbers of L2 cache misses per 1000 instructions (L2 MPKI). Applications whose L2 MPKI values are larger than ten are classified as MEM applications; the other applications are classified as ILP applications. Table VII shows nine four-core multi-programming workloads randomly selected from these applications. The MEM workloads consist of memory-intensive applications; and the ILP workloads contain compute-intensive applications. Then we randomly mix memory-intensive and compute-intensive applications as the MIX workloads to simulate cases that applications with different memory access behaviors are running together. For each SPEC2000 application, SimPoint 3.0 (79) is used to generate a simulation point of 100 million instructions. For each SPEC2006 application, the

Parameter	Value
Processor	4 cores, 3.2GHz, 4-issue per core, 16-stage pipeline
Functional units	4 IntALU, 2 IntMult, 2 FPALU, 1 FP-Mult
IQ, ROB and LSQ size	IQ 64, ROB 196, LQ 32, SQ 32
Physical register num	228 Int, 228 FP
Branch prediction	Hybrid, 8k global + 2k local, 16-entry RAS, 4k-entry and 4-way BTB
L1 caches (per core)	64KB Inst/64KB Data, 2-way, 64B line, hit latency: 1-cycle Inst/3-cycle Data
L2 cache (shared)	4MB, 4-way, 64B line, 15-cycle hit latency
MSHR entries	Inst: 8, Data: 32, L2: 64
Memory	1/2/4/8 channels, 1/2/4/8/16 DIMMs/channel, 2 ranks/DIMM, 8 banks/rank
Memory controller	64-entry buffer, 12ns overhead for scheduling, 15ns overhead for scheduling and timing
UDIC	4ns overhead for device timing
DDR3/UniMA channel bandwidth	800/1066/1333/1600 MT/s (Mega Transfers/second), 8 bytes/channel
DDR3 DRAM latency	DDR3-1066:8-8-8: precharge /row access /column access: 15ns DDR3-1600:11-11-11: precharge /row access /column access: 13.75ns
Pseudo-PCM parameter	Pseudo-PCM-1: 800Mhz data bus speed; column access: 15ns; row access: 66ns; precharge: 180ns Pseudo-PCM-2: 800Mhz data bus speed; column access: 15ns; row access: 250ns; precharge: 2us

TABLE VI: Major simulation parameters for UniMA.

Workload	Applications	Workload	Applications
MEM-1	swim,applu, art,lucas	ILP-1	vortex,gcc, six-track,mesa
MEM-2	fma3d,mgrid, galgel,equake	ILP-2	gromacs,namd, dealII,povray
MEM-3	milc,leslie3d, libquan-tum,lbm	ILP-3	povray,calculix, sjeng,omnetpp
MIX-1	lucas,equake, ammp,gap	MIX-2	gobmk,lbm, hmmer,gamess
MIX-3	leslie3d,sjeng, so-plex,omnetpp		

TABLE VII: Workload mixes for UniMA.

simulation is first fast-forwarded for one billion instructions, then the caches and memory systems are warmed up for 100 million instructions, and the detailed statistics are collected for the next 100 million instructions. The performance is characterized using weighted speedup (80), $\sum_{i=1}^{n}(IPC_{multi}[i]/IPC_{single}[i])$, where n is the total number of applications running, IPC$_{multi}$[i] is the IPC value of application i running under multi-core environment and IPC$_{single}$[i] is the IPC value of the same application running alone.

5.4 Experimental Results

In this section, we first evaluate and analyze the performance overhead of UniMA-based homogeneous memory systems. The results actually show that UniMA may improve performance when the memory system is under heavy load. Then, we evaluate and analyze the performance of UniMA-based heterogeneous memory systems.

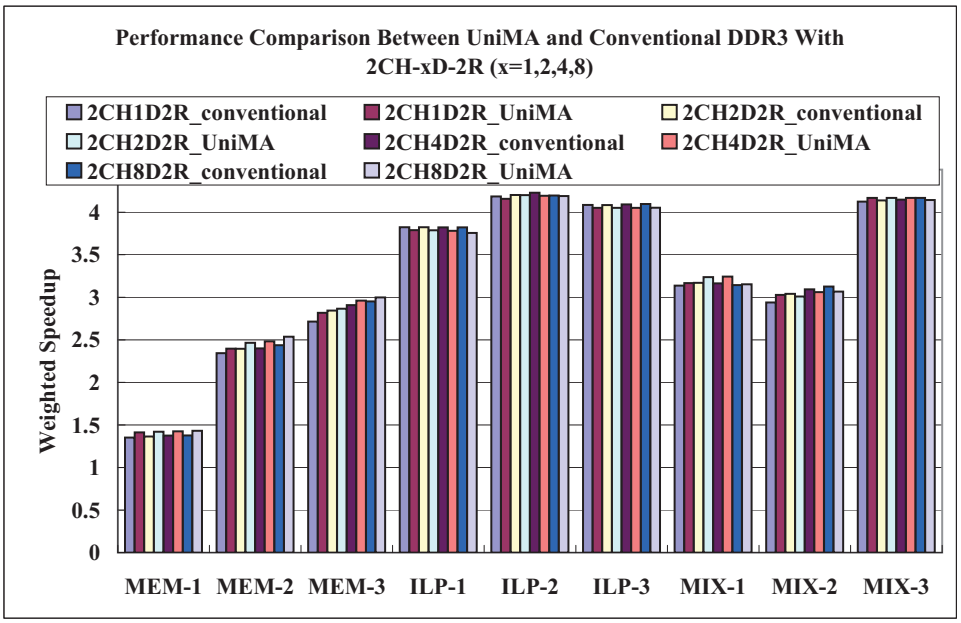


Figure 24: Performance comparison of UniMA and conventional DDR3-1066 system as the number of DIMMs per channel changes.

5.4.1 Performance with Homogeneous DDR3 Memory Devices

First, we will use our prototype UniMA design to show that a UniMA-based homogeneous memory system works efficiently when compared with the conventional DDR3 memory system, which has to be homogeneous. It represents the case that a UniMA-capable processor, which can work with many types of memory modules, is used with UniMA-based DDRx memory modules in a given system. We are interested in how much overhead that the UniMA may introduce. Throughout the experiments, we use xCH-yD-zR to represent a system configuration of x channels, y DIMMs per channel and z ranks per DIMM. The close page policy and XOR-based mapping (95; 45) are used if not mentioned otherwise.

Figure 24 compares the performance of UniMA systems with DDR3-1066 devices and conventional DDR3-1066 systems. The two different types of systems run at the same bus speed, and both have two channels and two ranks per DIMM. The number of DIMMs per channel varies from one to two, four and eight. The results show that, in general, the two types of systems have comparable overall performance.

For MEM workloads, the UniMA outperforms the DDR3 memory system by 0.8% to 4.5% (3.1% on average). The improvement comes from the additional parallelism at memory module level: multiple UDICs can concurrently issue device commands to their local devices. In the conventional DDR3 system, the time interval between read/write command and its data transfer is fixed. The centralized scheduler has to delay a command if that would cause a future bus conflict, which may delay other operations. This constraint is determined by the characteristics of centralized scheduler and is removed in UniMA. As shown in the figure, as the number of DIMMs per channel increases, the performance gain by the UniMA system increases, which indicates that the UniMA system can better utilize the extra module-level parallelism from the added DIMMs.

For MIX workloads, the UniMA system causes an average performance loss of 1.1% (from 0.6% to 1.8%) compared with the conventional DDR3 system. The performance penalty mainly comes from the UDIC buffer latency and the overhead introduced by token passing. As shown in the figure, as the number of DIMMs per channel increases, the performance of UniMA generally increases first and then decreases compared to that of conventional DDR3 system. This is a combined effect of both the increase of DIMM-level parallelism and the increase of

token passing overhead. Given that the buffer latency remains the same, the extra delay comes from the longer time to pass the token to a ready DIMM when more DIMMs are connected to a channel. This effect is not obvious for memory-intensive workloads because they tend to have many busy DIMMs and it is very likely that the token can reach a ready DIMM before the data bus becomes idle.

For ILP workloads, the performance degradation of UniMA compared to DDR3 ranges from 0.1% to 1.7% (0.8% on average). Although the ILP workloads may see more idle DIMMs and thus longer token passing overhead, their performance is also less affected by memory system. In addition, our prototype UniMA design includes a "Need Token" line to avoid unnecessary token passing operations.

5.4.2 Overhead of Token Passing Mechanism

In this section, we further investigate the overhead of token passing. To demonstrate the effect, we fix the number of DIMMs per channel to be sixteen and then vary the number of channels from one to two and four. Figure 25 shows the percentage of cycles that a token is held by a DIMM who doesn't need the token (but the bus could be busy) as p_t and a subset of that with an additional condition that the bus is free as p_t_df. Note that only the second case may cause potential performance loss and the delay is counted as the overhead of token passing. We can see that, in general, although the possibility that the token is passed to a DIMM without ready data is large (41.3% and 30.9% on average for MEM and MIX workloads, respectively), our scheme that begins the token passing during the data transfer can successfully hide the overhead. The possibility that the token cannot reach a DIMM with ready data when the data

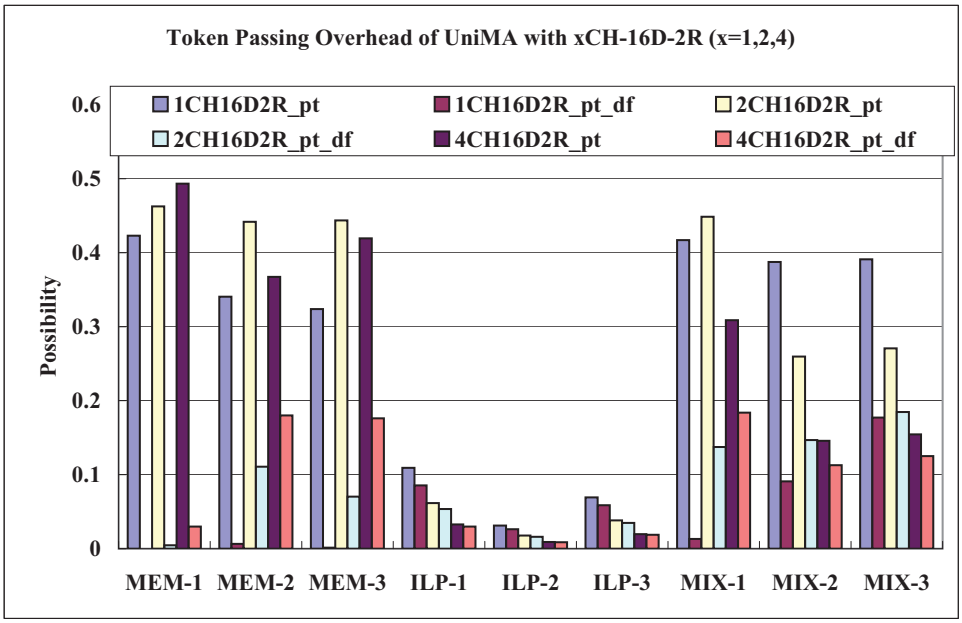


Figure 25: Token passing overhead of UniMA with 16 DIMMs per channel.

bus becomes free is only 6.4% for MEM and 13.0% for MIX workloads on average, respectively. The MIX workloads generate less memory traffic than the MEM workloads. Thus, the overhead of token passing cannot be hidden as well as for MEM workloads. This is the reason that we see larger performance loss on MIX workloads than on MEM workloads (1.1% loss vs. 3.1% gain on average). Additionally, as the number of channels increases, the value of P_t decreases for MIX workloads. This is because token passing is only triggered when another DIMM on the channel needs the token and sets the "Need Token" line; and more channels means lower possibility that the token passing is triggered for each channel. For the same reason, ILP workloads (with very light memory traffic) have very low P_t values.

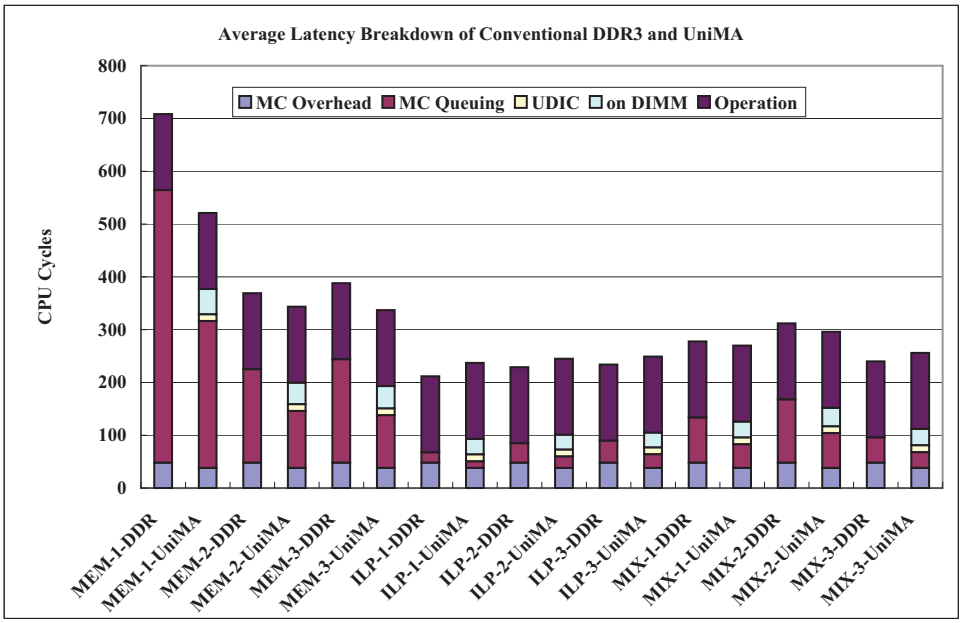


Figure 26: Latency breakdown of conventional DDR3 memory system and UniMA.

5.4.3 Memory Access Latency

Next, we will discuss the impact of using UniMA on the memory access latency. Figure 26 shows the latency breakdown for conventional DDR3 and UniMA systems. We use the same 2CH-4D-2R configuration and DDR3-1066 memory devices in the experiments. "MC Overhead" is the latency of memory controller scheduling; "MC Queuing" is the queueing delay in the controller; "UDIC" is the buffer and operation overhead of UDIC; "on DIMM" is the queueing delay on each DIMM, and "Operation" is the operation latency of memory devices. The "UDIC" and "on DIMM" overhead for the conventional DDR3 memory system is zero.

In general, the overall latency for MEM workloads is smaller under UniMA than under conventional DDR3 system (15.5% on average) due to the significant reduction on queueing delay. This is because using UDIC increases the DIMM-level parallelism; and the queueing delay on DIMM appears to be small because requests are distributed to each module. For MIX and ILP workloads, UniMA may slightly decrease (as 2.8% for MIX-1 and 5.1% for MIX-2) or slightly increase (as 8.0% on average for others) the overall latency. Because those workloads have fewer concurrent memory requests and thus shorter queueing delay under the conventional DDR3 system. The latency overhead of UDIC cannot be fully mitigated by the reduction on queueing delay.

5.4.4 Impact of Scheduling Choices on UDIC

As discussed in Section 5.2.3, the UDIC does not perform request reordering to minimize its cost and complexity. In this section, we evaluate the performance impact of this design choice by comparing it to an ideal scheduling scheme: the memory controller and all the UDICs perform request reordering as if the scheduling information is globally shared. Experiments are done for both close page and open page modes. The memory organization is set to 2CH-4D-2R with DDR3-1066 devices and 533MHz data bus. Figure 27 compares the performance with the two scheduling implementations. The two schemes, without and with the local reordering, yield almost the same performance (with up to 0.7% difference), for both the open and close page modes. The results indicate that, when there are sufficient numbers of DIMMs on each channel (four in this case), local request reordering is not necessary. Note that the memory controller can still perform certain priority-based request reordering.

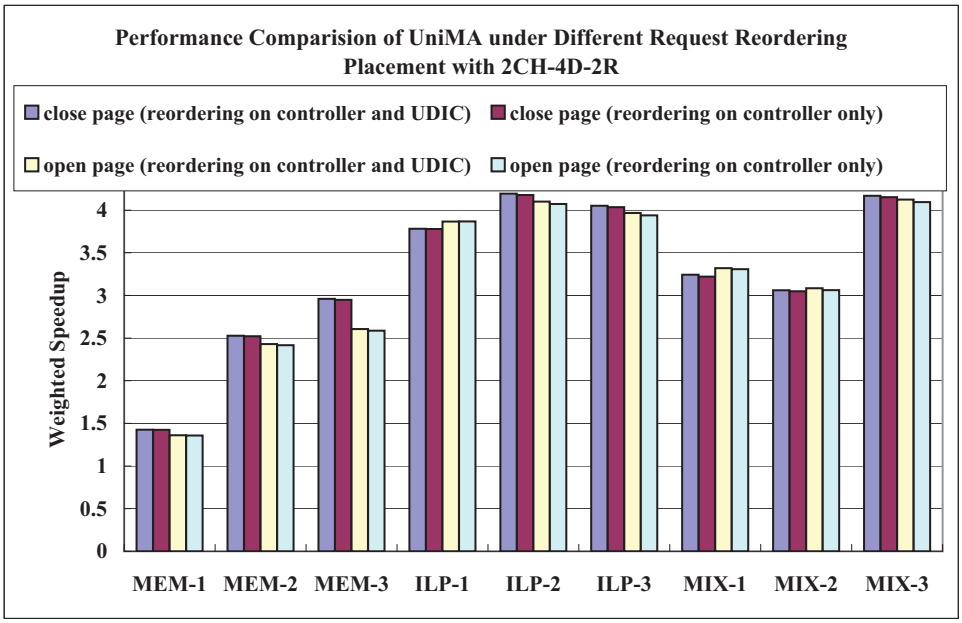


Figure 27: Performance comparison of UniMA without and with request reordering at UDICs.

5.4.5 UniMA with DDR3 and Pseudo-PCM Devices

A unique advantage of the UniMA framework is that it opens the door to building heterogeneous memory systems, i.e. with memory devices of different technologies, data rates and latencies in a single system; for example, a mix of PCM and DRAM memory modules. In this section, we evaluate and analyze how the prototype UniMA design performs in such a setting.

The experiments use the same 2CH-4D-2R configuration as before, except that now each channel has two DDR3-1600 DIMMs and two pseudo-PCM DIMMs. The pseudo-PCM devices have the same data rate as the DDR3-1600 devices, but the PCM storage core has much longer access latency than that of DRAM. A buffer may be put on each PCM DIMM to reduce its

average latency (and to improve write endurance), a common approach used in existing PCM studies (41; 69; 99). We call them pseudo-PCM modules because our simulation does not model low-level details except simple operations on PCM storage core and an optional, simple buffer. A real PCM module may involve complicated PCM operations and a complex buffer, whose details can be hidden from the memory controller by the UDIC in our design. The performance comparison includes the results with and without an 8MB buffer on each pseudo-PCM DIMM. The buffer works as a four-way set-associative cache with LRU replacement and a block size of 64 bytes. The hit latency is set to fifteen memory cycles. Note that a real PCM module may use a DRAM buffer of variant access latencies; in a conventional memory organization, the cache tags plus DRAM buffer status must reside in the memory controller for performing the centralized memory access scheduling.

As shown in Figure 28, the performance of a memory system of PCM1 modules (without buffer) is on average 25.4%, 18.7% and 4.9% lower than that of the DRAM system for MEM, MIX and ILP workloads, respectively. The performance of heterogeneous system with PCM1 and DRAM modules is only 16.0%, 13.2% and 3.2% lower than that of the DRAM system for those workloads, respectively. When the buffer is included on pseudo-PCM1 modules, the performance of the heterogeneous memory system is only 12.3%, 10.8% and 2.6% lower than that of the DDR3 system, for MEM, MIX and ILP workloads, respectively. The results for configurations using much slower pseudo-PCM2 device are similar. Note that we assume a simple interleaving scheme of mapping programs' memory pages evenly onto PCM and DRAM devices. A real system may employ a sophisticated mapping scheme, which is out of the scope

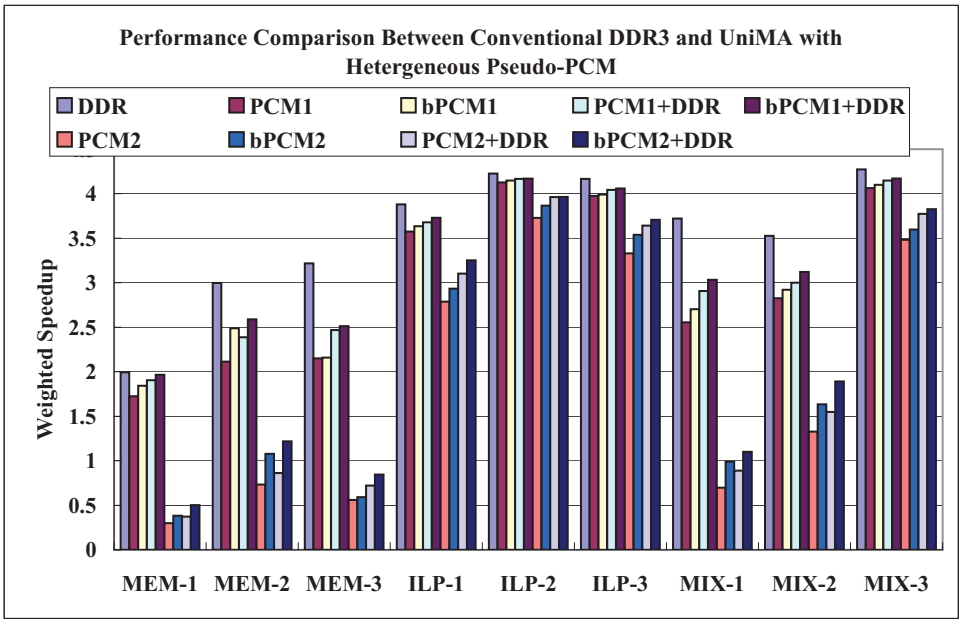


Figure 28: Performance comparison of traditional DDR3 system and UniMA with both pseudo-PCM and DDR3 DIMMs. (PCMx stands for pure PCM module; bPCMx stands for configuration of PCM module with an 8MB buffer).

of this study. Our intention here is to show that UniMA can support heterogeneous devices efficiently at architecture level.

5.5 Conclusion

We have proposed a memory architecture framework, with localized memory access scheduling, for universal interoperability between processors and memory modules. Such an architecture is critical to the adoption of emerging memory technologies in real systems. To evaluate the efficiency of such an architecture, which is a major concern at this time, we present a prototype implementation of Universal Memory Architecture (UniMA) over the DDRx memory channel. Our simulation results indicate that UniMA can be an efficient way of integrating different memory technologies, and localized memory scheduling can actually improve memory system efficiency for memory-intensive workloads. The potential of using heterogeneous memory modules inside a single system is particularly promising. The future research may include memory channel re-design under UniMA, further study on coordinated bus scheduling, UDIC design optimizations, memory modules with advanced cache structure, power optimization under UniMA, and utilization of a heterogeneous memory systems.

CHAPTER 6

CONCLUSION

With the wide spread use of multi-core and many-core processors, current memory system designers try to push the memory bandwidth and capacity to meet the user demands. A negative side-effect is the continuous increase on memory power consumption. In addition, main memory system design is severely limited by the rigid architecture that requires the memory controller to track the internal status of all memory devices (chips) and schedule the timing of all device operations. As a result, DRAM memory system is heading to the scalability wall. New memory technologies such as Phase-Change Memory (PCM) and STT-RAM emerge as potential alternatives to replace DRAM in future memory systems. Although those technologies have better energy-efficiency and scalability than DRAM, they also suffer from low write-endurance and long write-latency. Thus, new memory architectures are needed for supporting future memory systems and balancing among performance, energy-efficiency, capacity and lifetime.

To address the issue, we propose a systematic support for improving memory system efficiency at the architecture level by three steps. Firstly, a new DRAM scheduling algorithm called Delayed Row Activation is proposed to make the DRAM more energy-efficient by allowing memory ranks stay at a low-power mode longer if the data bus ownership cannot be acquired immediately after row activation finishes. Secondly, we present a heterogeneous mini-rank memory architecture that allows concurrently running applications to have different sub-rank

widths based on their memory access behavior. By dynamically assigning and changing the sub-rank configurations, the balance can be achieved between the performance and power saving, and large performance loss can be avoided. Lastly, we build a new memory architecture framework called Universal Memory Architecture (UniMA) that can support different memory technologies in a computer system by decoupling the scheduling of device operations from memory controller. A bridge chip is added to each memory module to perform device-specific scheduling locally.

Throughout this thesis, we demonstrate that our schemes can save DRAM power, provide optimal energy efficiency for mini-rank kind of design and integrate diverse memory technologies into one memory system with small overhead. A combination of the techniques in this thesis is straightforward and would be able to show a path to utilize both dominant and future memory technologies in a single computer system.

Our future work includes how to efficiently manage the hybrid memory systems containing diverse memory devices through scheduling and page mapping, and how to improve the write performance of new multi-level cell PCM (MLC-PCM) technology by allowing inaccurate values be stored into the MLC-PCM array and restoring the correct values later during memory read operations. Those are all important issues for successful transition of memory systems to the new era.

CITED LITERATURE

1. UMC free library. `http://freelibrary.faraday-tech.com/`.

2. Ahn, J. H., Jouppi, N. P., Kozyrakis, C., Leverich, J., and Schreiber, R. S.: Future scaling of processor-memory interfaces. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pages 1–12, New York, NY, USA, 2009.

3. Alameldeen, A., Chishti, Z., Wilkerson, C., Wu, W., and Lu, S.-L.: Adaptive cache design to enable reliable low-voltage operation. IEEE Transactions on Computers, 60(1):50–63, January 2011.

4. Ausavarungnirun, R., Chang, K. K.-W., Subramanian, L., Loh, G. H., and Mutlu, O.: Staged memory scheduling: achieving high performance and scalability in heterogeneous systems. In Proceedings of the 39th Annual International Symposium on Computer Architecture, pages 416–427, 2012.

5. Beamer, S., Sun, C., Kwon, Y.-J., Joshi, A., Batten, C., Stojanovic, V., and Asanovic, K.: Re-architecting DRAM memory systems with monolithically integrated silicon photonics. In Proceedings of the 37th Annual International Symposium on Computer Architecture, pages 129–140, 2010.

6. Binkert, N. L., Dreslinski, R. G., Hsu, L. R., Lim, K. T., Saidi, A. G., and Reinhardt, S. K.: The m5 simulator: Modeling networked systems. IEEE Micro, 26(4):52–60, 2006.

7. Bivens, A., Dube, P., Franceschini, M., Karidis, J., Lastras, L., and Tsao, M.: Architectural design for next generation heterogeneous memory systems. In 2010 IEEE International Memory Workshop (IMW), pages 1–4, 2010.

8. Burger, D., Goodman, J. R., and Kagi, A.: Memory bandwidth limitations of future microprocessors. In Proceedings of the 23rd International Symposium on Computer Architecture, pages 78–89, 1996.

9. Carter, J., Hsieh, W., Stoller, L., Swansony, M., Zhang, L., Brunvand, E., Davis, A., Kuo, C.-C., Kuramkote, R., Parker, M., Schaelicke, L., and Tateyama, T.: Impulse:

CITED LITERATURE (Continued)

Building a smarter memory controller. In Proceedings of the Fifth International Symposium on High-Performance Computer Architecture, pages 70–79, January 1999.

10. Caulfield, A. M., De, A., Coburn, J., Mollov, T. I., Gupta, R. K., and Swanson, S.: Moneta: A high-performance storage array architecture for next-generation, non-volatile memories. In 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 385–395, December 2010.

11. Cho, S. and Lee, H.: Flip-n-write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, pages 347–357, 2009.

12. Clinton W. Smullen, I., Mohan, V., Nigam, A., Gurumurthi, S., and Stan, M. R.: Relaxing non-volatility for fast and energy-efficient STT-RAM caches. In 17th International Conference on High-Performance Computer Architecture, pages 50–61, February 2011.

13. Cuppu, V. and Jacob, B.: Concurrency, latency, or system overhead: Which has the largest impact on uniprocessor DRAM-system performance? In Proceedings of the 28th International Symposium on Computer Architecture, pages 62–71, June 2001.

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

15. Delaluz, V., Kandemir, M., Vijaykrishnan, N., Sivasubramaniam, A., and Irwin, M. J.: DRAM energy management using software and hardware directed power mode control. In Proceedings of the 7th International Symposium on High-Performance Computer Architecture, pages 159–169, 2001.

16. Desikan, R., Lefurgy, C. R., Keckler, S. W., and Burger, D.: On-chip MRAM as a high-bandwidth, low-latency replacement for dram physical memories. Technical report tr-02-47, 2002.

17. Dhiman, G., Ayoub, R., and Rosing, T.: PDRAM: A hybrid pram and dram main memory system. In 2009 46th IEEE Design Automation Conference, pages 664–669, July 2009.

CITED LITERATURE (Continued)

18. Diniz, B., Guedes, D., Wagner Meira, J., and Bianchini, R.: Limiting the power consumption of main memory. In Proceedings of the 34th International Symposium on Computer Architecture, pages 290–301, 2007.

19. Ebrahimi, E., Lee, C. J., Mutlu, O., and Patt, Y. N.: Prefetch-aware shared resource management for multi-core systems. In Proceedings of the 38th Annual International Symposium on Computer Architecture, pages 141–152, June 2011.

20. Fan, X., Ellis, C., and Lebeck, A.: Memory controller policies for DRAM power management. In Proceedings of the 2001 International Symposium on Low Power Electronics and Design, pages 129–134, 2001.

21. Fan, X., Ellis, C., and Lebeck, A.: Modeling of DRAM power control policies using deterministic and stochastic Petri nets. In Proceedings of the 2nd international conference on Power-aware computer systems, pages 130–140, February 2002.

22. Fang, K., Chen, L., Zhang, Z., and Zhu, Z.: Memory architecture for integrating emerging memory technologies. In The Twentieth International Conference on Parallel Architectures and Compilation Techniques, pages 1–10, Oct. 2011.

23. Fang, K., Zheng, H., Lin, J., Zhang, Z., and Zhu, Z.: Mini-rank: A power-efficient ddrx dram memory architecture. IEEE Transactions on Computers, 99(PrePrints):1, 2012.

24. Fang, K., Zheng, H., and Zhu, Z.: Heterogeneous mini-rank: Adaptive, power-efficient memory architecture. In Proceedings of the 2010 39th International Conference on Parallel Processing, pages 21–29, 2010.

25. Fang, K. and Zhu, Z.: Conservative row activation to improve memory power efficiency. In Proceedings of the 27th international ACM conference on International conference on supercomputing, ICS '13, pages 81–90, New York, NY, USA, 2013. ACM.

26. Ganesh, B., Jaleel, A., Wang, D., and Jacob, B.: Fully-Buffered DIMM Memory Architectures: Understanding Mechanisms, Overheads and Scaling. In Proceedings of the 13th International Symposium on High-Performance Computer Architecture, pages 109–120, February 2007.

27. Ghosh, M. and Lee, H.-H. S.: Smart Refresh: An enhanced memory controller design for reducing energy in conventional and 3D Die-Stacked DRAMs. In Proceedings of

CITED LITERATURE (Continued)

the 40th International Symposium on Microarchitecture, pages 134–145, December 2007.

28. Guo, X., Ipek, E., and Soyata, T.: Resistive computation: avoiding the power wall with low-leakage, STT-MRAM based computing. In Proceedings of the 37th Annual International Symposium on Computer Architecture, pages 371–382, 2010.

29. Huang, H., Pillai, P., and Shin, K. G.: Design and implementation of power-aware virtual memory. In Proceedings of the USENIX Annual Technical Conference 2003 on USENIX Annual Technical Conference, pages 57–70, 2003.

30. Huang, H., Shin, K. G., Lefurgy, C., and Keller, T.: Improving energy efficiency by making DRAM less randomly accessed. In Proceedings of the 2005 International Symposium on Low Power Electronics and Design, pages 393–398, 2005.

31. Hur, I. and Lin, C.: Adaptive history-based memory schedulers. In Proceedings of the 37th International Symposium on Microarchitecture, pages 343–354, December 2004.

32. Hur, I. and Lin, C.: A comprehensive approach to DRAM power management. In Proceedings of the 13th International Symposium on High-Performance Computer Architecure, pages 305–316, 2008.

33. Platform 2015 - enterprise platform and integration concepts. White paper, INTEL Corporation, 2005.

34. The problem of power consumption in servers. Technical report, INTEL Corporation, 2012.

35. Intel, Inc.: Large-Capacity, High-Bandwidth Memory Solution. `http://cache-www.intel.com/cd/00/00/27/84/278436_278436.pdf`, 2006.

36. Joshi, M., Zhang, W., and Li, T.: Mercury: A fast and energy-efficient multi-level cell based phase change memory system. In 17th International Conference on High-Performance Computer Architecture (HPCA-17 2011), pages 345–356, February 2011.

37. Kaseridis, D., Stuecheli, J., and John, L. K.: Minimalist open-page: a DRAM page-mode scheduling policy for the many-core era. In Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, pages 24–35, 2011.

CITED LITERATURE (Continued)

38. Kim, Y., Han, D., Mutlu, O., and Harchol-balter, M.: ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. In Proceedings of the 16th International Symposium on High Performance Computer Architecture, pages 1–12, 2010.

39. Kim, Y., Papamichael, M., Mutlu, O., and Harchol-Balter, M.: Thread cluster memory scheduling: Exploiting differences in memory access behavior. In Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, pages 65–76, 2010.

40. Lebeck, A. R., Fan, X., Zeng, H., and Ellis, C.: Power aware page allocation. In Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 105–116, 2000.

41. Lee, B. C., Ipek, E., Mutlu, O., and Burger, D.: Architecting phase change memory as a scalable dram alternative. In Proceedings of the 36th annual international symposium on Computer architecture, pages 2–13, 2009.

42. Lee, H. G. and Chang, N.: Low-energy heterogeneous non-volatile memory systems for mobile systems. Journal of Low Power Electronics - JOLPE, 1(1):52–62, 2005.

43. Li, X., Li, Z., David, F., Zhou, P., Zhou, Y., Adve, S., and Kumar, S.: Performance directed energy management for main memory and disks. In Proceedings of the 11th International Conference on Architectural support for Programming Languages and Operating Systems, pages 271–283, 2004.

44. Lim, K., Chang, J., Mudge, T., Ranganathan, P., Reinhardt, S. K., and Wenisch, T. F.: Disaggregated memory for expansion and sharing in blade servers. In Proceedings of the 36th Annual International Symposium on Computer Architecture, pages 267–278, 2009.

45. Lin, W., Reinhardt, S. K., and Burger, D.: Reducing DRAM latencies with an integrated memory hierarchy design. In Proceedings of the Seventh International Symposium on High-Performance Computer Architecure, pages 301–312, January 2001.

46. Malladi, K. T., Nothaft, F. A., Periyathambi, K., Lee, B. C., Kozyrakis, C., and Horowitz, M.: Towards energy-proportional datacenter memory with mobile DRAM. In Proceedings of the 39th Annual International Symposium on Computer Architecture, pages 37–48, 2012.

CITED LITERATURE (Continued)

47. Mathew, B. K., McKee, S. A., Carter, J. B., and Davis, A.: Design of a parallel vector access unit for SDRAM memory systems. In Proceedings of the Sixth International Symposium on High-Performance Computer Architecture, pages 39–48, January 2000.

48. McKee, S. A., Wulf, W. A., Aylor, J. H., Salinas, M. H., Klenke, R. H., Hong, S. I., and Weikle, D. A. B.: Dynamic access ordering for streamed computations. IEEE Trans. Comput., 49(11):1255–1271, November 2000.

49. McKee, S. A. and Wulf, W. A.: Access ordering and memory-conscious cache utilization. In Proceedings of the First IEEE Symposium on High-Performance Computer Architecture, pages 253–262, January 1995.

50. McKee, S. A. and Wulf, W. A.: A memory controller for improved performance of streamed computations on symmetric multiprocessors. In Proceedings of the 10th International Parallel Processing Symposium, pages 159–165, April 1996.

51. MetaRAM, Inc.: MetaRAM product brief. `http://www.metaram.com/pdf/briefs/MetaRAM_DDR3_PB.pdf`.

52. Micron Technology, Inc.: DDR3 SDRAM System-Power Calculator. `http://download.micron.com/downloads/misc/ddr3_power_calc.xls`.

53. Micron Technology, Inc.: Double Data Rate (DDR) SDRAM Specification. `http://download.micron.com/pdf/misc/JEDEC79R2.pdf`, March 2003.

54. Micron Technology, Inc.: HTF18C64-128-256x72D. `http://download.micron.com/pdf/datasheets/modules/ddr2/HTF18C64_128_256x72D.pdf`, 2007.

55. Mishra, A. K., Dong, X., Sun, G., Xie, Y., Vijaykrishnan, N., and Das, C. R.: Architecting on-chip interconnects for stacked 3d STT-RAM caches in cmps. In Proceedings of the 38th Annual International Symposium on Computer Architecture, June 2011.

56. Mogul, J. C., Argollo, E. A., shah, M., and Faraboschi, P.: Operating system support for NVM+DRAM hybrid main memory. Technical report hpl-2009-256, May 2009.

57. Mutlu, O. and Moscibroda, T.: Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In Proceedings of the 40th International Symposium on Microarchitecture, pages 146–160, December 2007.

CITED LITERATURE (Continued)

58. Mutlu, O. and Moscibroda, T.: Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In Proceedings of the 35th Annual International Symposium on Computer Architecture, pages 63–74, 2008.

59. Nesbit, K. J., Aggarwal, N., Laudon, J., and Smith, J. E.: Fair queuing CMP memory systems. In Proceedings of the 39th International Symposium on Microarchitecture, pages 208–222, December 2006.

60. Nirschl, T., Philipp, J. B., Happ, T. D., Burr, G. W., Rajendran, B., Lee, M. H., Schrott, A., Yang, M., Breitwisch, M., Chen, C. F., Joseph, E., Lamorey, M., Cheek, R., Chen, S.-H., Zaidi, S., Raoux, S., Chen, Y., Zhu, Y., Bergmann, R., Lung, H.-L., and Lam, C.: Write strategies for 2 and 4-bit multi-level phase-change memory. 2007 IEEE International Electron Devices Meeting, pages 461–464, Dec. 2007.

61. Pandey, V., Jiang, W., Zhou, Y., and Bianchini, R.: DMA-aware memory energy management. In Proceedings of the 12th International Symposium on High-Performance Computer Architecture, pages 133–144, February 2006.

62. Papandreou, N., Pozidis, H., Mittelholzer, T., Close, G., Breitwisch, M., Lam, C., and Eleftheriou, E.: Drift-tolerant multilevel phase-change memory. In 3rd IEEE International Memory Workshop (IMW), pages 1–4, May 2011.

63. Patel, A., Afram, F., Zeng, H., and Ghose, K.: MARSSx86 - micro-architectural and system simulator for x86-based systems. `http://www.marss86.org/index.php/Home`.

64. Patounakis, G., Li, Y. W., , and Shepard, K. L.: A fully integrated on-chip DC-DC conversion and power management system. Solid-State Circuits, 39(3):443–451, 2004.

65. Phadke, S. and Narayanasamy, S.: MLP aware heterogeneous memory system. In Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 1–6, March 2011.

66. Qureshi, M. K., Franceschini, M., and Lastras, L.: Improving read performance of phase change memories via write cancellation and write pausing. In Proceedings of the 16th International Symposium on High Performance Computer Architecture, pages 1–11, 2010.

CITED LITERATURE (Continued)

67. Qureshi, M. K., Franceschini, M. M., Lastras-Montano, L. A., and Karidis, J. P.: Morphable memory system: A robust architecture for exploiting multi-level phase change memories. In Proceedings of the 37th Annual International Symposium on Computer Architecture, pages 153–162, 2010.

68. Qureshi, M. K., Karidis, J., Franceschini, M., Srinivasan, V., Lastras, L., and Abali, B.: Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, pages 14–23, 2009.

69. Qureshi, M. K., Srinivasan, V., and Rivers, J. A.: Scalable high performance main memory system using phase-change memory technology. In Proceedings of the 36th annual international symposium on Computer architecture, pages 24–33, 2009.

70. Rambus, Inc.: Rambus Unveils Next Generation XDR Memory Interface. `http://www.rambus.com/us/news/press_releases/2005/050707.html`, 2005.

71. Raoux, S., Burr, G. W., Breitwisch, M. J., Rettner, C. T., Chen, Y.-C., Shelby, R. M., Salinga, M., Krebs, D., Chen, S.-H., Lung, H.-L., and Lam, C. H.: Phase-change random access memory: A scalable technology. IBM Journal of Research and Development, 52(4/5):465–479, July/September 2008.

72. Rixner, S.: Memory controller optimizations for web servers. In Proceedings of the 37th International Symposium on Microarchitecture, pages 355–366, December 2004.

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

74. Samsung, Inc.: Samsung Ships Industry's First Multi-chip Package with a PRAM Chip for Handsets. `http://www.samsung.com/us/aboutsamsung/news/newsIrRead.do?news_ctgry=irnewsrelease&page=1&news_seq=18828&rdoPeriod=ALL&from_dt=&to_dt=&search_keyword=`, April 2010.

75. Schechter, S., Loh, G. H., Strauss, K., and Burger, D.: Use ECP, not ECC, for hard failures in resistive memories. In Proceedings of the 37th Annual International Symposium on Computer Architecture, pages 141–152, 2010.

76. Schrom, G., Hazucha, P., Hahn, J., Gardner, D. S., Bloechel, B. A., Dermer, G., Narendra, S. G., Karnik, T., and De, V.: A 480-mhz, multi-phase interleaved buck DC-

CITED LITERATURE (Continued)

DC converter with hysteretic control. In IEEE 35th Annual Conference on Power Electronics Specialists, volume 308, pages 4702–4707, 2004.

77. Seong, N. H., Woo, D. H., Srinivasan, V., Rivers, J., and Lee, H.-H.: SAFER: Stuck-at-fault error recovery for memories. In 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, pages 115–124, December 2010.

78. Shao, J. and Davis, B. T.: A burst scheduling access reordering mechanism. In Proceedings of the 13th International Symposium on High-Performance Computer Architecture, pages 285–294, February 2007.

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

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

81. Standard Performance Evaluation Corporation: SPEC CPU2006. http://www.spec.org.

82. Sun, G., Dong, X., Xie, Y., Li, J., and Chen, Y.: A novel architecture of the 3d stacked MRAM L2 cache for CMPs. In Proceedings of the 15th International Symposium on High Performance Computer Architecture, pages 239–249, 2009.

83. Sun, G., Joo, Y., Chen, Y., Niu, D., Xie, Y., Chen, Y., and Li, H.: A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement. In Proceedings of the 16th International Symposium on High Performance Computer Architecture, pages 1–12, 2010.

84. Synopsys Corp.: Synopsys design compiler. `http://www.synopsys.com/products/logic/design_compiler.html`.

85. Udipi, A., Muralimanohar, N., Chatterjee, N., Balasubramonian, R., Davis, A., and Jouppi, N.: Rethinking DRAM design and organization for energy-constrained multi-cores. In Proceedings of the 37th Annual International Symposium on Computer Architecture, pages 175–186, 2010.

CITED LITERATURE (Continued)

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

87. Wilkerson, C., Alameldeen, A. R., Chishti, Z., Wu, W., Somasekhar, D., and Lu, S.-L.: Reducing cache power with low-cost, multi-bit error-correcting codes. In Proceedings of the 37th Annual International Symposium on Computer Architecture, pages 83–93, 2010.

88. Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R., and Xie, Y.: Hybrid cache architecture with disparate memory technologies. In Proceedings of the 36th Annual International Symposium on Computer Architecture, pages 34–45, 2009.

89. Xu, W., Liu, J., and Zhang, T.: Data manipulation techniques to reduce phase change memory write energy. In Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design, pages 237–242, 2009.

90. Yoon, D. H., Chang, J., Muralimanohar, N., and Ranganathan, P.: Boom: Enabling mobile memory based low-power server DIMMs. In Proceedings of the 39th Annual International Symposium on Computer Architecture, pages 25–36, 2012.

91. Yoon, D. H. and Erez, M.: Virtualized and flexible ECC for main memory. In Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 397–408, 2010.

92. Yoon, D. H., Muralimanohar, N., Chang, J., Ranganathan, P., Jouppi, N., and Erez, M.: FREE-p: Protecting non-volatile memory against both hard and soft errors. In 17th International Conference on High-Performance Computer Architecture, pages 466–477, February 2011.

93. Zhang, L., Fang, Z., Parker, M., Mathew, B. K., Schaelicke, L., Carter, J. B., Hsieh, W. C., and McKee, S. A.: The Impulse memory controller. IEEE Transactions on Computers, 50(11):1117–1132, November 2001.

94. Zhang, W. and Li, T.: Characterizing and mitigating the impact of process variations on phase change based memory systems. In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, pages 2–13, 2009.

CITED LITERATURE (Continued)

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

96. Zheng, H., Lin, J., Zhang, Z., Gorbatov, E., David, H., and Zhu, Z.: Mini-rank: Adaptive DRAM architecture for improving memory power efficiency. In Proceedings of the 41st International Symposium on Microarchitecture, pages 210–221, November 2008.

97. Zheng, H., Lin, J., Zhang, Z., and Zhu, Z.: Decoupled DIMM: building high-bandwidth memory system using low-speed DRAM devices. In Proceedings of the 36th International Symposium on Computer Architecture, pages 255–265, June 2009.

98. Zheng, H. and Zhu, Z.: Power and performance trade-offs in contemporary dram system designs for multicore processors. IEEE Transactions on Computers. accepted.

99. Zhou, P., Zhao, B., Yang, J., and Zhang, Y.: A durable and energy efficient main memory using phase change memory technology. In Proceedings of the 36th Annual International Symposium on Computer Architecture, pages 14–23, 2009.

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

101. Zhu, Z., Zhang, Z., and Zhang, X.: Fine-grain priority scheduling on multi-channel memory systems. In Proceedings the Eighth International Symposium on High-Performance Computer Architecture, pages 107–116, February 2002.

VITA

EDUCATION

Ph.D. in Computer Engineering 08/2007 – 08/2013
Department of Electrical and Computer Engineering (ECE),
University of Illinois at Chicago (UIC), United States

M.E. in Computer Architecture 09/2005 – 05/2007
School of Computer Science and Technology (CS)
Huazhong University of Science and Technology (HUST), China

B.E. in Computer Science 09/2001 – 05/2005
School of Computer Science and Technology (CS)
Huazhong University of Science and Technology (HUST), China

B.E. in Communication Engineering 09/2001 – 05/2005
School of Computer Science and Technology (CS)
Huazhong University of Science and Technology (HUST), China

PUBLICATIONS

Journal Papers

- **Kun Fang**, Hongzhong Zheng, Jiang Lin, Zhao Zhang, Zhichun Zhu, "Mini-Rank: A Power-Efficient DDRx DRAM Memory Architecture," IEEE Transactions on Computers, 02 Oct. 2012. IEEE computer Society Digital Library. IEEE Computer Society, http://doi.ieeecomputersociety.org/10.1109/TC.2012.240

Conference Papers

- **Kun Fang** and Zhichun Zhu, "Conservative Row Activation to Improve Memory Power Efficiency", The 27th International Conference on Supercomputing (ICS), Pages 81–90, Eugene, Oregon, June 10-14, 2013
- **Kun Fang**, Long Chen, Zhao Zhang and Zhichun Zhu, "Memory Architecture for Integrating Emerging Memory Technologies", The 20th International Conference on Parallel Architectures and Compilation Techniques (PACT), Pages 403–412, October 10-14, 2011.
- **Kun Fang**, Hongzhong Zheng and Zhichun Zhu, "Heterogeneous Mini-rank: Adaptive, Power-Efficient Memory Architecture", In Proceedings of the 2010 International Conference on Parallel Processing (ICPP-2010), Pages 21–29, San Diego, CA, September 13-16, 2010.

PATENTS

- Hai Jin, Zhiyuan Shao, **Kun Fang**, Shi Luo and Huacai Chen, "A Parallel Multi-CPU Virtual Machine", China Patent ZL200710168720.9, issued December 23, 2009.

RESEARCH EXPERIENCES

Research Assistant with Prof. Zhichun Zhu 08/2007 – 06/2013
Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago

- *Recent Research*: **Conservative row activation** 06/2012 - 03/2013
  - Delay activation to precharged rank to let it stay in low power mode longer to save power if data bus is busy.
- *Recent Research*: **Dynamic heterogeneous mini-rank** 09/2011 - 05/2012

VITA (Continued)

- Investigate how to dynamically manage and assign heterogeneous mini-rank configuration based on application bandwidth requirement.
- Provide a hardware/software support for efficiently utilize heterogeneous mini-rank design.
- Port memory simulation module to MARSSx86 simulator.

• *Past Research*: **Universal memory architecture** 01/2010 - 03/2011
- Build a Universal Memory Architectural framework to support different memory technologies in one system.
- Implement the proposed architecture framework and integrated into our memory module.
- Cross compile SPEC2006 alpha binary on fedora 11 box.
- Build detailed memory simulation module for M5 2.0 simulator.

• *Past Research*: **Heterogeneous mini-rank** 11/2008 - 04/2010
- Making different mini-rank configurations co-exist in a memory system to obtain near optimal power efficiency of memory system.
- Assigning difference memory configurations to applications to reduce memory power while maintaining performance.
- Implement the proposed architecture and integrated into M5 1.0 simulator.

Research Assistant with Prof. Zhiyuan Shao 09/2005 – 05/2007
Key Lab of Services Computing Technology and System, the Ministry of Education (SCTS)
Key Lab of Cluster and Grid Computing Lab, the Hubei Province (CGCL)
School of Computer Science and Technology (CS)
Huazhong University of Science and Technology (HUST), China
• *Project*: **Parallelization of Boches functional simulator**
- Making each simulated CPU of Boches simulator into threads and execute in parallel.

TEACHING EXPERIENCE

Teaching Assistant *ECE267*: **Computer Organization I** Fall 2007, Spring 2008
• Computer organization; assembly language programming; memory, CPU and I/O organization; programming techniques and tools; programming laboratory.
Teaching Assistant *ECE367*: **Microprocessor Based Design** Fall 2008
• Microprocessor architecture; micro programmed machines; control signals and timing; system buses; parallel and serial interfacing; interrupt processing.

WORKING EXPERIENCE

GPU Architecture Intern *NVIDIA* Summer 2011
• Working on a detailed GPU simulator in memory system team.
• Implement functions for coordinating last level cache and frame buffer. Implement new memory controller functions for supporting GDDR5 memory device.

AWARDS

- Memory Scheduling Championship, Energy Track. 3rd JILP Workshop on Computer Architecture Competitions (JWAC-3), 2012
- Excellent Undergraduate Student, HUST, 2005