공학박사 학위논문

Architecting Main Memory Systems of Manycore Processors

매니코어 프로세서 시스템을 위한 주 메모리 시스템 아키텍처 설계

2015 년 8 월

서울대학교 융합과학기술대학원

융합과학부 지능형융합시스템전공

오 성 일

Architecting Main Memory Systems of Manycore Processors

매니코어 프로세서 시스템을 위한 주 메모리 시스템 아키텍처 설계

지도 교수 안 정 호

이 논문을 공학박사 학위논문으로 제출함
2015 년 7 월

서울대학교 융합과학기술대학원
융합과학부 지능형융합시스템전공
오 성 일

오성일의 공학박사 학위논문을 인준함
2015 년 7 월

위 원 장 박 재 흥 (인)

부위원장 안 정 호 (인)

위 원 곽 노 준 (인)

위 원 김 동 준 (인)

위 원 이 재 욱 (인)

Abstract

Architecting Main Memory Systems of Manycore processors

Seongil O

Intelligence Systems

Department of Transdisciplinary Studies

The Graduate School

Seoul National University

Manycore processors have already become mainstream, where DRAM is widely used as main memory for these manycore processor systems. Applications have also been parallelized to exploit manycore systems efficiently, and their data sets keep increasing. Therefore, main memory systems become the performance and energy bottleneck of modern manycore systems. Through-silicon interposer (TSI) technology is a promising solution to architect high bandwidth energy-efficient main memory systems for modern manycore processors. While TSI improves the I/O energy efficiency, it results in an "unbalanced" memory system design because DRAM core dominates the overall energy consumption of manycore systems. However, there are few studies

on DRAM device microarchitecture that consider the system-level impact on the performance and energy efficiency of manycore systems.

To conduct research on modern manycore systems, we need a cycle-level timing simulator that provides the detailed microarchitecture models of core and uncore subsystems. The core subsystems of manycore processors can consist of traditional or asymmetric cores. The uncore subsystems become more powerful and complex than ever, including deeper cache hierarchies, advanced on-chip interconnects, memory controllers, and main memory. We first implement a new cycle-level timing simulator, McSimA+, which enables microarchitectural studies on manycore systems and have the detailed microarchitecture models of core and uncore subsystems. McSimA+ is an application-level+ simulator, which enjoys the light weight of application-level simulators and the full control of threads and processes as in full-system simulators.

Then, we evaluate the system-level impacts on the performance and power of DRAM array organizations. We model modern DRAM array organizations by varying the number of banks, DRAM row size per bank, and evaluate the area, power, and timing of them. The modeling results show that larger DRAM row improves area efficiency and access time, but increases activation/precharge energy. We evaluate the system-level impacts of DRAM array organizations by simulating a manycore system with 3D stacked DRAM memory. The system performance and energy

efficiency improve as each DRAM rank has more banks. While the 8KB DRAM row shows the best performance, the highest energy-delay product (EDP) is obtained when the DRAM row size is 2KB.

We finally propose a new TSI-based main memory system which solves the "unbalance" between I/O energy and DRAM core energy. Our TSI-based main memory system utilizes a novel DRAM device microarchitecture, called *μbank*. The μbank partitions each conventional bank into a large number of smaller banks (or μbanks) that operate independently with minimal area overhead. A massive number of μbanks provide ample bank-level parallelism, less bank conflict rate, and thus improve both IPC and EDP by 1.62× and 4.80× respectively for memory intensive SPEC 2006 benchmarks on average over the baseline DDR3-based memory system. We also show that the μbank-based memory systems can simplify memory controller designs because they show comparable performance with simple open-page policy to complex prediction-based page management policies. In our μbank-based memory system, the simple open-page policy achieves more than 95% of the performance of a perfect predictor.

Keywords : manycore, memory system, DRAM, μbanks, manycore simulator, Through-Silicon Interposer (TSI)
Student Numbers : 2009-23807

Contents

Abstract i

Contents v

List of Figures ix

List of Tables xi

1 Introduction 1

1.1 Research Contributions 6

1.2 Outline 9

2 Main Memory System Organizations 10

2.1 DRAM Microarchitecture 16

2.2 DRAM Modules or DRAM Microarchitectures for Manycore Systems 20

2.2.1 New DRAM Modules for Manycore Systems 21

2.2.2 New DRAM Microarchitectures for Manycore Systems 23

2.3 Memory Controller Organizations 27
2.3.1 Memory Access Scheduling Policies 28
2.3.1 Page Management Policies 32

3 Manycore Simulation Infrastructure 35

3.1 Why Yet Another Simulator? 39

3.2 McSimA+: Overview and Operation 43
3.2.1 Thread Management for Application-level+ Simulation 46
3.2.2 Implementing the Thread Management Layer 48

3.3 Modeling of Manycore Architecture 50
3.3.1 Modeling of Core Subsystem 51
3.3.2 Modeling of Cache and Coherence Hardware 55
3.3.3 Modeling of Network-on-Chips (NoCs) 60
3.3.4 Modeling of the Memory Controller and Main Memory . 61

3.4 Validation 62

3.5 Clustering Effect in Asymmetric Manycore Processors 69
3.5.1 Manycore with Asymmetric Within or Between Clusters 69
3.5.2 Evaluation 72

3.6 Limitations and Scope of McSimA+ 77

4 Energy-Efficient DRAM Array Organizations 79

4.1 Energy-Efficient DRAM Array Organizations 81

4.2 Evaluation 86
4.2.1 Experimental Setup 86
4.2.2 The System-level Impact of DRAM Array Organizations 87

5 Silicon Interposer-Based Main Memory Systems..... 90

5.1 Through-Silicon Interposer (TSI) 94

5.1.1 TSI Technology Overview..... 94

5.1.2 The Energy Efficiency and Latency Impact of the TSI. 97

5.2 Microbank: A DRAM Device Organization for TSI-based Main Memory Systems. 100

5.2.1 Motivation for Microbank..... 100

5.2.2 Microbank Overhead 105

5.3 Revisiting DRAM Page Management Policies 108

5.4 Evaluation 110

5.4.1 Experimental Setup..... 110

5.4.2 Test Workloads 112

5.4.3 The System-level Impact of the μbanks..... 114

5.4.4 The Impact of Address Interleaving and Prediction Based Page-Management Schemes on μbanks 118

5.4.5 The Impact of OS Page Migration 121

5.4.6 The Impact of DRAM Refresh 122

5.4.7 The Impact of TSI on Processor-Memory Interfaces 124

5.5 Related Work 126

6 Conclusion 130

6.1 Future Work 132

Bibliography 135

국문초록 149

감사의 글 153

List of Figures

Figure 2.1 A conventional main memory system. 12
Figure 2.2 DRAM power breakdown. 12
Figure 2.3 The block diagram of a modern memory controller. ... 13
Figure 2.4 Conventional DRAM organization. 16
Figure 2.5 Contemporary DRAM operations. 18
Figure 2.6 A memory system with Multicore DIMMs. 21
Figure 2.7 Subarray-level parallelism (SALP). 23
Figure 2.8 Half-DRAM 25
Figure 2.9 Selective subarray access (SSA) 26
Figure 2.10 Memory access scheduling. 28
Figure 2.11 Parallelism-aware batch scheduling (PAR-BS) 30
Figure 3.1 McSimA+ infrastructure. 44
Figure 3.2 Example manycore architecture models. 50
Figure 3.3 Core modeling in McSimA+. 52
Figure 3.4 Cache coherence microarchitecture modeling. 56
Figure 3.5 The relative IPC of simulation results normalized to that of native machines. 64
Figure 3.6 Validation of L2 cache simulation results. 66
Figure 3.7 Clustered manycore architectures. 68
Figure 3.8 Mixed workloads used in the case study. 73

Figure 3.9 Performance comparison between SWAB and AWSB architectures. 73
Figure 4.1 Overall layout of the exemplar system. 82
Figure 4.2 A distributed bank and page layout of a DRAM rank. . 83
Figure 4.3 Power, area, and timing results of 4Gb main-memory DRAM bank modeled by the modified CACTI. 85
Figure 4.4 Average of the normalized IPC and EDP of the tested workloads on a chip-multiprocessing system. 88
Figure 5.1 Energy breakdown of the conventional PCB-based, TSI-based, and μbank-based memory system. 92
Figure 5.2 Packaging technologies. 94
Figure 5.3 μbank organization and operations. 103
Figure 5.4 An example of a μbank design. 104
Figure 5.5 The relative DRAM area and energy while varying the number of partitions in the BL and WL directions. .. 105
Figure 5.6 A manycore system with 16 clusters..... 111
Figure 5.7 The relative IPC of 429.mcf, spec-high, and TPC-H. 115
Figure 5.8 The relative 1/EDP of 429.mcf, spec-high, and TPC-H. 115
Figure 5.9 The relative IPC, 1/EDP, and power breakdown of applications on representative μbank configurations. 116
Figure 5.10 Two of the possible address interleaving schemes... 118
Figure 5.11 The IPC, relative EDP, and row-buffer miss rate of the representative μbank configurations..... 122
Figure 5.12 The performance overhead and DRAM refresh power of the representative μbank configurations. 123
Figure 5.13 The IPC, power, and relative EDP of 3 processor-memory interfaces. 125

List of Tables

Table 2.1 Comparison of high-density memory technologies.... 11
Table 2.2 Row buffer (RB) conflict rate of individual applications in SPEC CPU2006 benchmark suite. 33
Table 3.1 Summary of existing simulators..... 38
Table 3.2 Configuration specifications of the validation target... 63
Table 3.3 Parameters including area and power estimations from McPAT. 68
Table 5.1 DRAM energy and timing parameters..... 98
Table 5.2 The SPEC CPU2006 applications are categorized into 3 groups depending on MAPKIs. 113

Chapter 1

Introduction

Modern manycore processors integrate more than tens of cores. As process technology enables billions of transistors to be integrated in a single chip, the core and uncore subsystems of modern manycore processors become more powerful and complex. The core subsystems can consist of conventional symmetric cores or asymmetric cores, such as Tile64 [4] that embeds 64 symmetric in-order cores into a single chip. Asymmetric multicore architecture, such as ARM big.LITTLE [5], forms a cluster with high performance out-of-order (OoO) Coretex-A15 cores and energy efficient in-order Cortex-A7 cores. The uncore subsystems of manycore processors include deeper cache hierarchies, scalable on-chip interconnects, multiple memory controllers, and main memory.

In this dissertation, we focus on architecting high performance memory systems for modern manycore processors. DRAM is

widely used as main memory for manycore processors because it provides larger storage density than SRAM and less random access latency than non-volatile memories. The primary goal of DRAM technology is to increase the storage density of DRAM chips. As a result, the data transfer rate and the random access latency of DRAM chips improves more slowly. Recent applications have been parallelized to exploit the maximum performance of manycore systems. These applications execute multiple tasks concurrently, and their memory access pattern becomes more random due to inter-thread interference. Consequently, the random access performance of main memory becomes more important [6] [7], and the memory system limits the performance of computer systems significantly in the manycore era. Moreover, the data sets of applications keep increasing. For example, recent in-memory databases [8] uses more than hundreds of gigabytes of main memory to cache frequently used data. Consequently, main memory DRAM consumes a significant portion of total energy consumed by manycore systems, forcing DRAM to become energy bottleneck as well [6].

The bandwidth of main memory systems can be improved by either increasing the number of pins or the data rate of each pin. On the conventional printed circuit board (PCB)-based main memory systems, however, increasing pin counts increases fabrication cost and energy consumption for off-chip data transfer. Increasing the data rate of each pin degrades energy efficiency and signal integrity. Through-silicon interposer (TSI) technology is an attractive

solution to address these problems [9] [10]. TSI-based packaging enables the implementation of high bandwidth energy-efficient processor-memory interfaces without increasing die area or off-package access energy by connecting a processor die and multiple 3D stacked DRAM dies on the interposer through low impendence wires and Through-silicon vias (TSVs). As the I/O energy of TSI-based memory systems decreases, the energy consumed by DRAM cores dominates the overall energy consumption of main memory systems. To address this problem, we should study DRAM core microarchitecture to architect more energy-efficient TSI-based memory systems.

A manycore simulation infrastructure is one of the challenges to conduct our study. Simulators have been used to validate the innovative ideas of researchers who study computer architecture. Because simulators enable researchers to save much time and money in prototyping their ideas, various simulators have been developed for their own purposes. For studies on manycore systems, high-level abstraction simulators are not appropriate because they usually provide overly simplified microarchitecture models for fast simulation speed. Full-system simulators, which run OSes, are relatively slow and sometimes require modifying OSes to support new technologies, such as ARM big.LITTLE. The modification of OSes becomes overhead when researchers do not focus their research on system/OS events. Therefore, we need a middle ground approach between high-level abstract simulators and full-system simulators. The new manycore simulation

infrastructure should provide a lightweight, flexible, and detailed microarchitecture-level simulation for emerging manycore systems.

To this end, we implement a manycore simulation infrastructure, called McSimA+, which provides fast simulation speed and the detailed microarchitecture models of components constructing manycore systems. McSimA+ supports detailed microarchitecture -level modeling not only of the cores, such as out-of-order, in-order, multithreaded, and single-threaded core, but also of all uncore components, including multi-level caches, NoCs, cache coherence hardware, memory controllers, and main memory DRAM. Moreover, innovative architecture designs, such as asymmetric manycore architecture, and 3D stacked main-memory systems are supported.

Then, we investigate the system-level impacts of various DRAM array organizations for the performance and energy efficiency. We evaluate the area, power, and timing of DRAM array organizations by varying the number of banks and DRAM row size of each bank. To evaluate the system-level impacts of these array organizations, we use McSimA+ to simulate a manycore processor with 3D stacked main memory. Our modeling results show that larger DRAM page lowers die area and access latency, but increases activate/ precharge energy. The simulation results show that a larger number of banks improve both the system-level performance and energy efficiency. Our tested system achieves the best performance with 8KB DRAM row but the best energy-delay product with 2KB DRAM row.

Based on the knowledge of DRAM array organizations and modern manycore system architectures, we propose a novel TSI-based main memory system that addresses the "unbalance" between I/O energy and DRAM core energy. The proposed main memory system utilizes a high-performance and energy-efficient DRAM device microarchitecture, called *μbank*. The μbank DRAM partitions each bank both horizontally and vertically into a large number of smaller banks (or μbank) which operate independently like conventional banks and reduces DRAM row access energy by activating fewer bits per activation. Furthermore, our evaluation show that simple open-page policy [11] provides comparable performance to sophisticated prediction-based page-management policies because more μbanks provides more open rows and lowers bank conflict rate. Therefore, μbank can simplify DRAM controller design. We also revisit the appropriate granularity of address interleaving and show that DRAM row interleaving outperforms cache-line interleaving because inter-threads interference mostly disappears with a massive number of μbanks.

1.1 Research Contributions

In summary, this dissertation makes the following contributions:

- McSimA+ models x86 based (asymmetric) manycore (up to more than 1,000 cores) microarchitectures in detail for both core and uncore subsystems, including a full spectrum of asymmetric cores (from single-threaded to multithreaded and from in-order to out-of-order), cache hierarchies, coherence hardware, NoC, memory controllers, and main memory. McSimA+ is an application-level+ simulator, representing a middle ground between a full-system simulator and an application-level simulator. Therefore it enjoys the light weight of an application-level simulator and full control of threads and processes as in a full-system simulator. It is flexible in that it can support both execution-driven and trace-driven simulations. McSimA+ enables architects to perform detailed and holistic research on manycore architectures.

- The validations of McSimA+ cover different processor configurations from the entire multicore processor to the core and uncore subsystems. The validation targets are comprehensive ranging from a real machine to published results. In all validation experiments, McSimA+ demonstrates good performance accuracy.

- The proposed asymmetric manycore design, Asymmetry Within a cluster and Symmetric Between clusters (AWSB), mitigates the thread migration overhead in modern asymmetric manycore systems. Using McSimA+, our study show that the AWSB design performs noticeably better than the state-of-art clustered asymmetric architecture as adopted in ARM big.LITTLE.

- The exploration of the system-level impact of modern DRAM array organizations shows that increasing the number of banks improves the performance and energy efficiency of manycore systems. Larger DRAM row improves area efficiency and access time but increase the energy consumed by activate and precharge commands.

- The μbank, a novel DRAM device organization for TSI-based memory systems, reduces the DRAM row activate and precharge energy while increasing bank-level parallelism. Therefore, the μbank realizes the full potential of the increased bandwidth and capacity offered by the TSI technology.

- A massive number of μbanks reduces the complexity of memory controllers because a complex DRAM page-management policy is not needed. We evaluate a novel

prediction-based DRAM page-management scheme which improves performance by up to 20.5% for a system without μbanks; however, the performance improvement over a simple open-page policy is limited to 3.9% with μbanks.

- Compared to a baseline system with DDR3-based processor-memory interfaces, our TSI-based μbank system improves performance (IPC) by 1.62x and energy-delay product by 4.80x on average for a third of the SPEC CPU2006 benchmarks with high main memory bandwidth demands.

1.2 Outline

We introduce the organization of this dissertation as follows.

In Chapter 2, we explain modern DRAM architecture and memory controller organizations. Chapter 3 describes the detailed organization and operations of our proposed manycore simulation infrastructure, called *McSimA+*. We show the advantages of McSimA+ as a simulation infrastructure for modern manycore system researches. We explore contemporary DRAM array organizations in Chapter 4, where we also evaluate the system-level impacts on the performance and energy efficiency of DRAM array organizations.

Chapter 5 describes μbank, a new DRAM device organization for a high-performance energy-efficient Through-Silicon Interposer (TSI)-based main memory system. We describe the advantages and limitations of TSI technology. We then quantify the energy and performance benefits of the TSI technology when it is applied to the main memory system, compared with the conventional DIMM-based memory system. We explain the detailed microarchitecture of μbank DRAM and show the system-level impacts of our μbank-based main memory system. Finally, Chapter 6 presents the conclusion of this dissertation.

Chapter 2

Main Memory System Organizations

The importance of the main memory system in a modern computer becomes more significant because memory footprints of modern applications and the number of cores sharing a memory controller keep increasing. In a stored-program digital computer, the central processing unit (CPU) of a computer fetches instructions and data from memory, execute them, and write processed results back to the memory. Therefore, the performance and energy efficiency of the computer system largely depend on its memory system. Ideally, a perfect memory can deal with every request instantaneously and hold all data of currently running processes, but it cannot be implemented in reality. Alternatively, modern computer systems place multi-level caches between a CPU and the main memory system to mimic the ideal memory. A cache is small and fast associative data storage but does not have its own address space.

	SRAM	DRAM	Flash	PCM
Size	100−200 F^2	6−8 F^2	~4 F^2	4−20 F^2
MLC support	No	No	Yes	Yes
Volatile	Yes	Yes	No	No
Read time	<1 ns	5−100 ns	10−100 ns	10−100 ns
Write time	<1 ns	5−100 ns	>1 ms	>100ns
Write energy	0.1 nJ/b	0.1 nJ/b	0.1−1 nJ/b	<1 nJ/b
Endurance	∞	∞	10^4−10^5	10^8

Table 2.1: Comparison of high−density memory technologies [12].

They hold copies of frequently accessed data in the main memory and provide fast memory access to the CPU. Although processor manufacturers have increased the levels of cache hierarchy and the capacity of a last level cache (LLC) to improve performance by reducing main memory accesses, main memory systems still act as the performance and energy−efficiency bottleneck of modern computer systems. Modern main memory systems consist of multiple main storage devices, memory controllers, and channels (Figure 2.1). Dynamic random access memory (DRAM) is used as the storage devices for main memory systems due to its high storage density although it is slower than static random access memory (SRAM) in random access latency (Table 2.1). A memory controller is connected with multiple dual−inline memory modules (DIMMs) through an offchip multi−drop bus, called a channel.

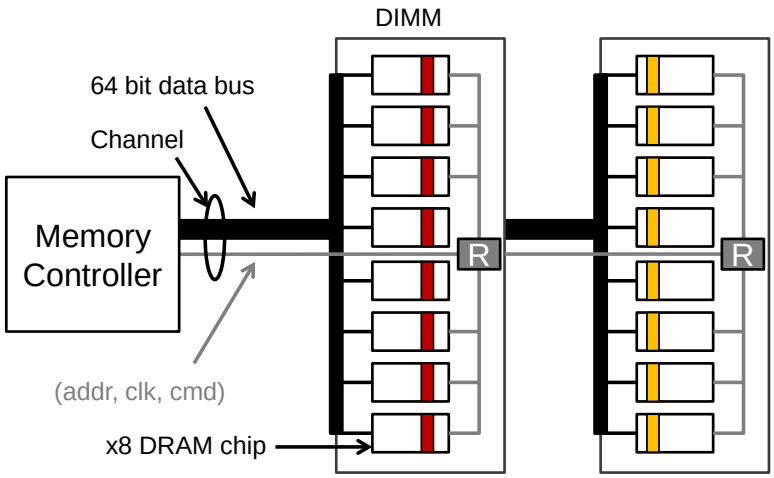


Figure 2.1: A conventional main memory system. A memory controller communicates with multiple DIMMs through a channel. A DIMM can have multiple ranks. A rank consists of multiple DRAM chips that operate in unison. In this example, eight x8 DRAM chips forms a 64 bit wide rank, and a cache line for read or write is distributed across all DRAM chips in the rank.

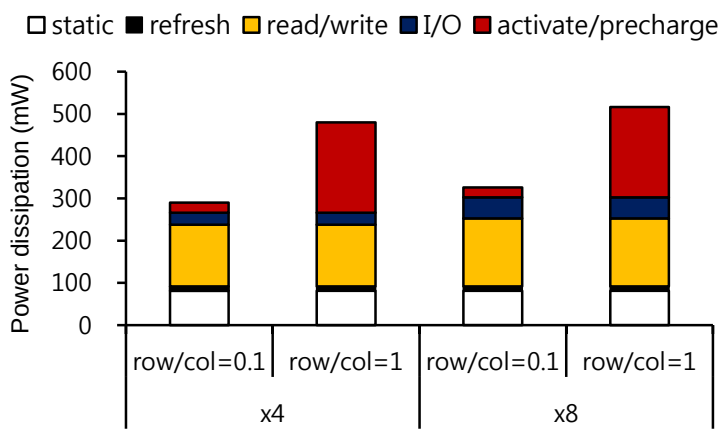


Figure 2.2: DRAM power breakdown of Micron 2 Gb DDR3 DRAM chip [13]. row/col is a ratio between ACTIVATE/PRECHARGE and READ/WRITE [6].

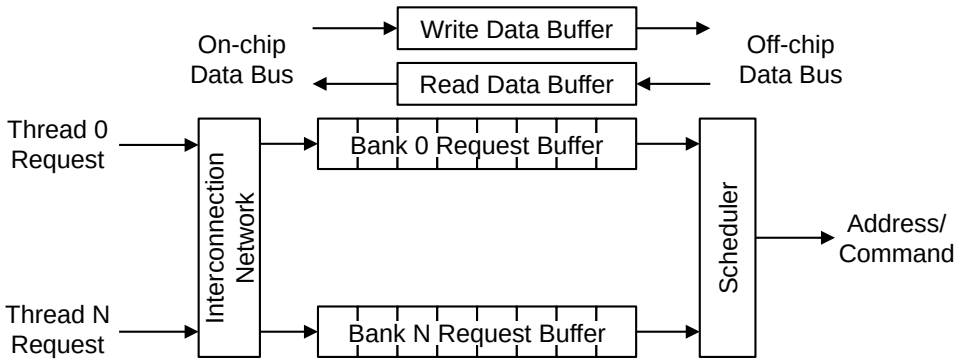


Figure 2.3: The block diagram of a modern high performance memory controller.

Each DIMM has more than one DRAM rank. A rank is a set of DRAM chips which are simultaneously controlled by the memory controller. The memory channel width is generally 64 bits or 72 bits if error collecting code (ECC) is included, but the external data I/O (DQ) width of a DRAM chip is only 4, 8, or 16 bits. Therefore, multiple DRAM chips which meet the width of channel form a rank, and they receive the same commands to operate in unison.

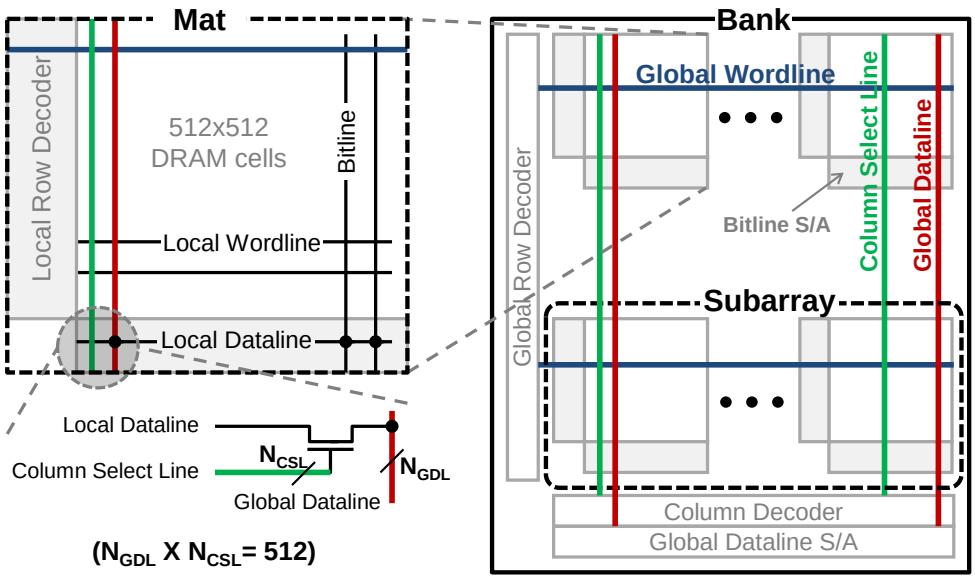
Continuous DRAM device scaling has enabled a single commodity DRAM chip to hold several gigabits. However, the latency of DRAM chips has been improved slowly, for the primary design goal of them is to lower manufacturing costs by improving area efficiency. To reduce manufacturing costs, unlike processor manufacturers, DRAM manufacturers use fewer metal layers, and wires which have higher resistance and capacitance. A DRAM chip consists of multiple 2-dimensional banks that hold storage cells. All banks in the DRAM chip share the external I/O but operate

independently of each other, which provides bank-level parallelism. Consequently, the performance of a main memory system can be improved by exploiting this bank-level parallelism, and more banks provide more bank-level parallelism. Each bank has multiple rows, each of which consists of several kilobits of DRAM cells. Since only one row can be accessed in a single bank at a time, to access data in a different row, the bank precharges its bitlines and activates an entire row of the cells storing the data belong to. The DRAM row access is the critical path of a DRAM access because it not only takes long time but also consumes large energy. The number of bits affected by a single activation is often 100 times or more than those of a cache line [14] [15]. These structural characteristics enhance area efficiency of DRAM chips, and the cost of activations and precharges can be amortized by multiple accesses to the activated row (Figure 2.2). Because modern manycore processor concurrently executes multiple processes, and their memory accesses become more random due to inter-thread interference, memory systems act as performance and energy-efficiency bottleneck in modern manycore systems.

Because this precharge activation takes dozens of nanoseconds, modern memory controllers concurrently utilize multiple banks by reordering and pipelining memory requests. Figure 2.3 shows the block diagram of a modern high performance memory controller. It has per-bank request buffers, each of which can have multiple outstanding memory requests toward its bank. When there are multiple pending memory requests, the scheduler

selects one of them, which is the most beneficial based on its scheduling policy. The scheduling policies generally improve system bandwidth and energy efficiency by exploiting the spatial locality of memory requests in the request queues. Thread-unaware scheduling policies [11] [119] have no intention to service each thread's in parallel. Therefore, they unfairly service threads with low row-buffer locality by prioritizing threads with high row-buffer locality. Recently, some advanced memory controllers consider the affinities between memory requests and their owner threads, and they generally use this information to improve fairness of service among threads [20] [26] [27] [109]. Together with scheduling policies, when there is no pending memory request toward a bank having an active row, memory controllers speculatively decide whether the bank keeps activated or not by their page-management policies. Miss prediction causes significant performance and energy penalty. If an active row is not precharged due to mis-prediction, the row will be precharged and activated to process incoming row-buffer hit requests. Therefore, Page-management policies also take charge of a significant portion of the performance and energy efficiency of memory systems, and they are highly affected by the memory access patterns of applications.

We describe the modern DRAM microarchitecture in Section 2.1. We summarize researches on DRAM modules and DRAM microarchitectures for manycore systems in Section 2.2. In Section 2.3, we summarize modern memory access scheduling policies and page-management policies.



(a) (b)

Figure 2.4: Conventional DRAM organization. (a) A two dimensional array called mat is a unit of storage structure and (b) the mats compose again a two-dimensional structure called bank where all the mats share a global row decoder and a column decoder (S/A = Sense Amplifiers).

2.1 DRAM Microarchitecture

Main-memory DRAM organization has evolved to improve throughput per device (die or package) while lowering random access latencies under a tight budget constraint in die area and fabrication complexity. A DRAM cell, which stores a bit of data, consists of an access transistor and a capacitor. DRAM cells are arranged in a two-dimensional array such that all the cells in a row

This Section is based on [1]. - ©[2014]IEEE Reprinted, with permissions, from SC' 14.

are controlled by a wordline (WL) and all the cells in a column share a datapath bitline (BL).

This wordline and bitline sharing improves area efficiency but also increases the capacitance and resistance of the datapath and control wires so that single-level drivers and sense amplifiers become inadequate, especially for modern multi-Gbit DRAM devices. Therefore, hierarchical structures [16] are leveraged in the datapath and control wires so that a long wordline is divided into multiple sub-wordlines, each having a dedicated driver and spanning a single mat. A mat is typically composed of 512×512 DRAM cells in a modern DRAM device; it is the building block for the DRAM structure as shown in Figure 2.4(a). The sub-wordlines within a mat are referred to as local wordlines. Similarly, bitlines refer to the datapath wires within a mat while the wires that are perpendicular to the wordlines and traversing an entire array to transfer data are called global datalines (Figure 2.4(b)). Local datalines, which are parallel with wordlines, connect bitlines and global datalines. A modern DRAM device has several of these arrays called banks. Datapath wires that encompass all the banks within a device are called inter-bank datalines.

Figure 2.5 show the contemporary DRAM operations. To access data in a DRAM device, a series of commands are applied. First, the row decoder decodes the address that is issued along with an activate command (ACT) and drives a particular global wordline. Each mat has a local row decoder that combines the signals from global wordlines and drives up to one local wordline.

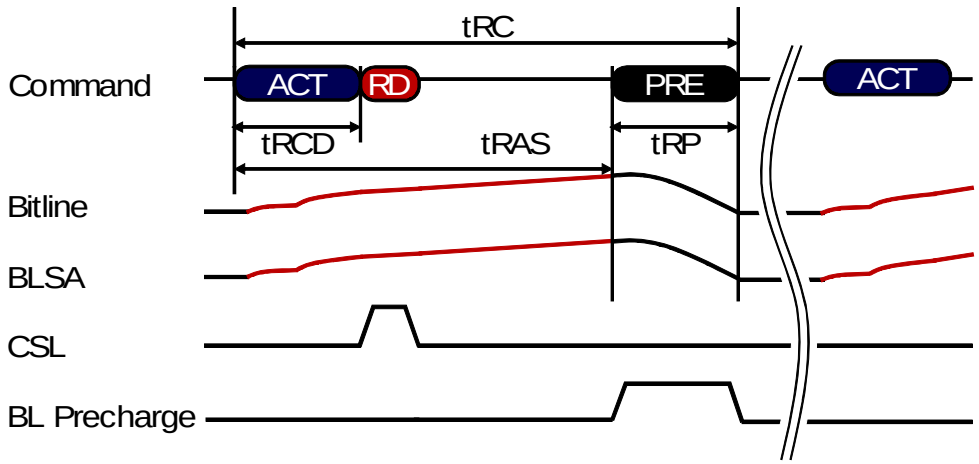


Figure 2.5: Contemporary DRAM operations. Bitline, bitline sense amplifier (BLSA), column select line (CSL), and bitline (BL) precharge show their voltage developments by each DRAM operations.

Therefore, a row of mats specified by the address activates the target local wordlines. All the transistors controlled by the local wordlines become open and share charges with the corresponding bitlines. A bitline sense amplifier is connected to each bitline because the capacitance of a bitline is much higher than that of a cell [17]. Therefore, a small voltage difference developed by the charge sharing must be amplified. These bitline sense amplifiers latch data and then restore those values back to the open (connected) cells because DRAM cells are passive.

Next, the column decoder drives the column select lines, which are parallel with global datalines, specified by one or more read commands (RDs) following the activate command after a minimum time of tRCD. The column select lines choose one or more bitlines

per mat to move data from the bitline sense amplifiers to the edge of the mat through local datalines. Each local dataline is connected to a global dataline that traverses the entire column of the bank. The global dataline also has a sense amplifier to reduce data transfer time because it is lengthy (spans a few millimeters) and connected to dozens of local datalines, one per row of mats called a subarray, in modern devices.

Once the data transfer is over, a precharge command (PRE) precharges the datapath wires to make them ready for the following transfers, which takes tRP. A write (WR) process is the same as a read except that the data transfer path is reversed. Note that these DRAM commands expose a portion of bitline sense amplifiers as row buffers so that they can be exploited in scheduling commands to the DRAM devices. When a device does not have enough bandwidth or capacity, multiple devices are grouped and operate in tandem forming a rank. Ranks are connected through a bus and controlled by a memory controller. The datapath width of a dual in-line memory module (DIMM), which hosts few ranks, is 64 bits.

Modern DRAM devices have multiple banks [18] to improve the data transfer rate on random accesses with little spatial locality. It takes up to a few nanoseconds (5ns on a DDR3-1600 DIMM [19]) for a DRAM rank to transfer a cache line, whose size is typically around 64 bytes. However, the time to activate a row, restore it, and precharge the datapath wires in a device, which is called the cycle time (tRC), is still around 50ns [19]. Unless an activated row is used for dozens of cache-line transfers, the datapath out of the

device would be greatly under-utilized. In order to alleviate this problem, mats in a device are grouped into multiple banks, each of which operates independently except that all banks in a channel share command and datapath I/O pads that are used to communicate to the outside of the DRAM die.

2.2 DRAM Modules or DRAM Microarchitectures for Manycore Systems

A manycore processor concurrently executes multiple tasks, and all the cores in the processor share on-chip main memory controllers. Memory accesses from different sources are gathered in a single memory controller, and they interfere with each other because each process has different address space. Because the inter-thread interference reduces the spatial locality of memory accesses, random access performance of memory systems becomes more important. The negative effect of the inter-thread interference can be alleviated through either hardware or software techniques. In the hardware perspectives, the impact of the inter-thread interference on performance and energy efficiency highly depends on DRAM module organizations, DRAM microarchitectures, and scheduling policies of main memory controllers. In this section, we mainly focus on the researches on DRAM module organizations and DRAM microarchitecture, and we explain the scheduling policies in Section 2.3.

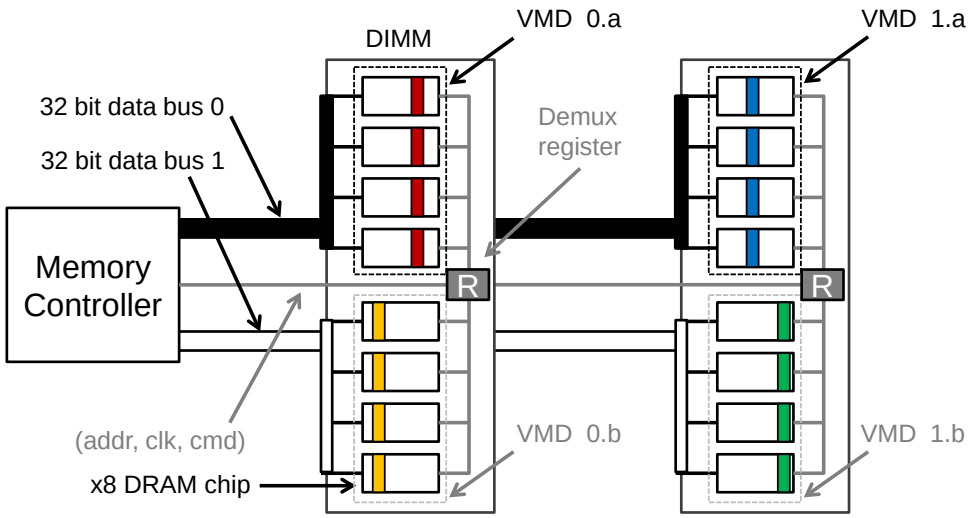


Figure 2.6: A memory system with Multicore DIMMs. Each DIMM is divided into two sub-ranks, which are called virtual memory devices (VMDs). The demux register routes address and control signals to each VMDs [6].

2.2.1 New DRAM Modules for Manycore Systems

The inter-thread interference also incurs overfetch and fairness problem among threads. In a manycore system, an active DRAM row is likely precharged after several row accesses because memory accesses become more random. The row size of recent DDR3 or DDR4 DRAM modules is generally 8KB, which is over 100x of a cache line. Therefore, too many bits are activated and precharged without being used, and this phenomenon is called overfetch [6] [7]. Since the fairness problem is highly related to the scheduling policy of a memory controller, we will revisit this in Section 2.3.

Rank subsetting, a memory module-level technique to reduce the overfetch problem, divides a DRAM rank into multiple sub-ranks which operates independently of each other. If a rank is divided into N subsets, the number of effective bank increases as many as N times, and the row size of each bank decreases as much as N times. As reduction in row size decreases the energy consumed by row accesses, it improves the energy efficiency of memory systems. Although the rank subsetting increases the serialization latency to transfer a cache line through a subset memory channel, it improves both the throughput and energy efficiency of memory systems because it also improves more bank-level parallelisms.

Multicore DIMM and Mini-rank are the implementations of the rank subsetting [6] [7]. Multicore DIMM divides a rank into multiple subsets with a demux register which routes or demultiplexes commands signals, address, and data signals (Figure 2.6). Mini-rank DIMM [7] also uses demuxs registers, but it uses them differently from Multicore DIMM. The key difference between two implementations is how they use their demux registers. Mini-rank DIMM uses demux registers to communicate command signals, data, address with a memory controller, whereas Multicore DIMM uses them only to communicate address and command signals. In addition, Mini-rank DIMM requires a demux register per rank subset, but Multicore DIMM requires only one demux register per memory channel. Therefore, Mini-rank DIMM needs more components and consumes more energy than Multicore DIMM.

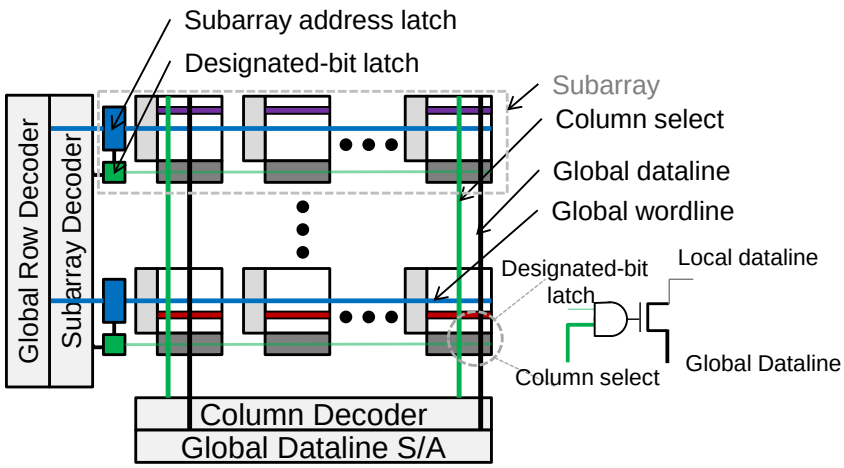


Figure 2.7: Subarray-level parallelism (SALP). Subarray address latches can hold the row address for each subarray and allow multiple rows to be activated across different subarray. Designated-bit latch connects local datalines of a selected subarray to the global datalines, which prevents the data collision among subarrays [95].

2.2.2 New DRAM Microarchitectures for Manycore Systems

Modern DRAM chips employ multi-bank architecture to process multiple memory requests in parallel. Unfortunately, if two memory requests attempt to access different rows in a bank, they must be serialized due to bank or row-buffer conflict. Since modern manycore processors expose multiple independent memory requests into their shared memory controller, the memory system stalls more often as it suffers from row-buffer conflicts more often. The row-buffer conflict rate is highly related to the number of available banks in memory systems and memory access scheduling

policies. In this section, we introduce multiple proposals that increase the number of banks in a DRAM chip.

Kim et al. proposed a new DRAM microarchitecture that exploits sub-array level parallelism (SALP) in a DRAM bank [95]. A subarray is a group of mats in a bank, which share global wordlines and operate in unison. As we describe in Section 2.1, in a bank, only one subarray can have an active row because all subarrays share the global row decoder and global bitlines. Therefore, the active row in the subarray must be precharged to activate a different row in a different subarray. To address this problem, SALP makes each subarray operate independently like the conventional bank by adding a subarray address latch and a designated-bit latch (Figure 2.7) to each subarray. The subarray address latch holds a row address to be activated in the subarray, and the designated-bit latch prevents data from other subarrays from colliding with shared global datalines in their bank. SALP improves bank-level parallelism by increasing the number of rows concurrently activated in a DRAM chips, but it cannot eliminate the waste of DRAM row access energy due to overfetch because the row size does not change regardless of the bank count.

Half-DRAM redesigned the structure of DRAM mats to reduce the energy consumption of row accesses [118]. In Half-DRAM, the conventional mat is split into two identical sub-mats which are called left and right (Figure 2.8). The row decoder drives the left and right sub-mats that originally belong to two neighbor mats.

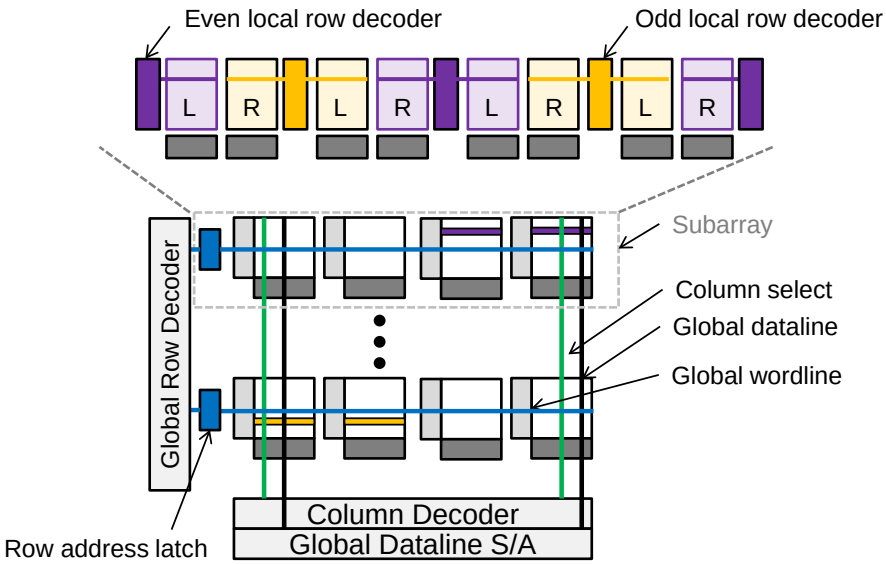


Figure 2.8: Half-DRAM. Every mat in a subarray is split into two sub-mats, left and right, and the subarray is divided into even and odd groups which can operate independently with their row address latch [118].

When the row decoder selects a wordline, its left and right sub-mats are selected, and the activated row size becomes the half. Although the bank is divided into logically two parts, odd and even, the bank can activate only one row out of two parts at a time. This problem can be addressed by adding local row address latches like SALP. Udipi el al. proposed DRAM architectures which eliminate the overfetch problem [71]. The first one is selective bitline activation (SBA) which activates small portion of the row in a subarray by dividing row with multiple sub-rows with sub-wordlines. In this design, a sub-row cannot be activated with row access strobe (RAS) before column access strobe (CAS) arrives.

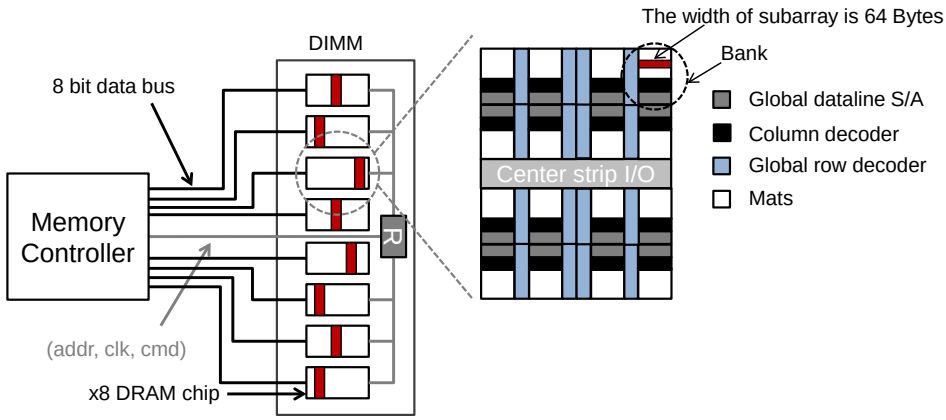


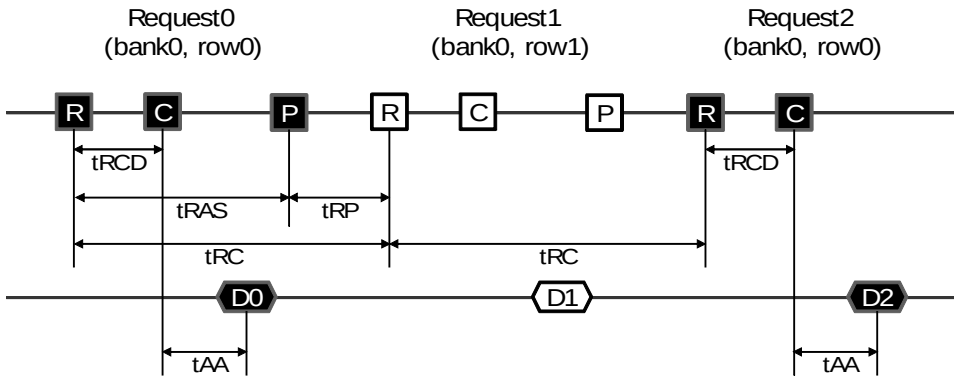
Figure 2.9: Selective subarray access (SSA). SSA eliminates the overfetch problem by reducing the width of subarray to the cache line size of the last level cache, 64 bytes. A rank is split into 8 sub-ranks, and each DRAM chip in a rank transfers a whole cache line [71].

In spite of an additional delay due to posted-RAS, SBA saves substantial power. Unfortunately, it requires complete redesign of the conventional DRAM architecture. The second one is selective subarray access (SSA) which utilizes both rank subsetting and the modification of DRAM microarchitecture (Figure 2.9). In this design, each DRAM chip in a DRAM module operates as a single rank. Inside a DRAM chip, a bank consists of multiple subarrays whose row size is exactly the same as the cache line size. Unlike the SBA design, a channel is also divided into multiple sub-channels, and each of them services a different cache line. Consequently, SSA provides much higher concurrency than SBA, but it increases the serialization latency due to reduction of channel width.

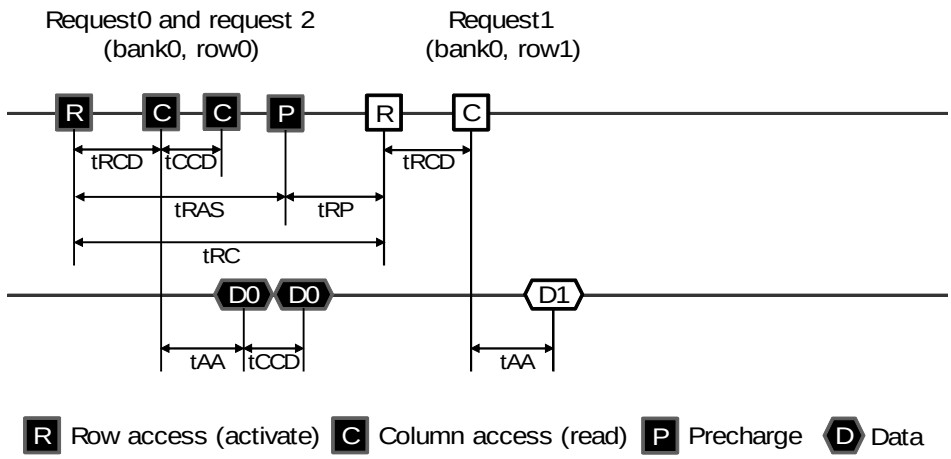
2.3 Memory Controller Organizations

In this section, we summarize the organizations of modern memory controllers. More complete description is found in related publications [11] [20] [21] [22]. Modern manycore processors often integrate more than one memory controller which acts as an interface between caches and the main memory DRAM. Modern high performance memory controllers consist of a scheduler, per-bank request buffers, a write buffer, and a read buffer (Figure 2.3). The scheduler is an important component of a memory controller, which reorders pending memory requests in its request queues in order to improve both latency and bandwidth.

Page-management policies, a part of memory access scheduling policies, determine when to deactivate an active DRAM page in case there is no pending request toward the active row in a request buffer. Most conventional schedulers generally choose one between two static page-management policies, open-page or close-page policy. The open-page policy keeps the recently accessed row activated (open) unless there is a pending request to the different row of the bank that includes the just accessed row. On the contrary, the close-page policy deactivates (closes) the accessed DRAM page when there is no pending memory request toward the open row. Recent studies proposed more sophisticated page-management policies to mitigate the performance penalty due to miss-predictions [23] [24] [25], which are also implemented in the memory controllers of modern server [24] [25].



(a) FCFS (in-order) scheduling



(b) FR-FCFS (out-of-order) scheduling [11]

Figure 2.10: Memory access scheduling.

2.3.1 Memory Access Scheduling Policies

The scheduler of a memory controller sends DRAM commands based on its memory access scheduling policy that manages the flow of data in a memory system. As shown in Figure 2.10, how a memory controller schedules DRAM commands largely affects the performance of a memory system. For example, three memory

requests, such as request0, request1, and request2, are buffered in order in the request queue of a memory controller. Request0 and request1 incur row buffer conflict. However, request0 and request2 can be processed consecutively in time because they access the same row, which means a row-buffer hit. In Figure 2.10(a), first-come-first-serve (FCFS) scheduler processes request in order. Because this scheduler cannot be aware of the row buffer locality between request0 and request2, it takes 2 x tRC + tRCD + tAA to complete the three requests. However, the first-ready FCFS (FR-FCFS) scheduler processes request2 prior to request1, and it only needs tRC + tRCD + tAA to complete the same three requests (Figure 2.10(b)). Even if FCFS is easy to implements, it is inefficient due to low channel utilization.

FR-FCFS considers all pending requests in request buffers in order to send a command to the bank that is ready to process the command. It starts from the oldest request. If the target bank of the request is not ready or busy serving another request, the scheduler pursues issuing a command to other banks to proceed, thus increasing throughput. When multiple threads send memory requests to the shared memory system, the FR-FCFS scheduling policy can make threads with less row buffer locality starved unfairly because it tends to prioritize the threads which higher row buffer locality to improve channel utilization. To address this problem, thread-aware memory access scheduling policies have been proposed [20] [26] [27].

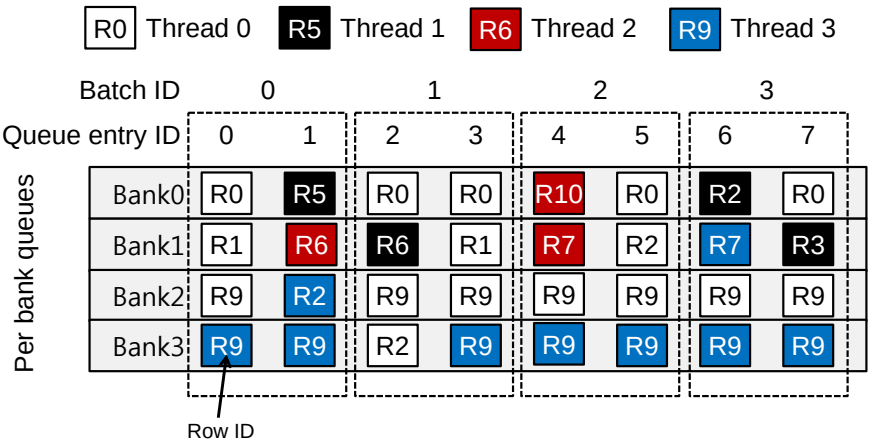


Figure 2.11: Parallelism-aware batch scheduling (PAR-BS). PAR-BS forms a batch of requests according to their arrival order. It ensures the worst-case response delay because the PAR-BS scheduler forms a new batch after processing all requests in the current batch [20].

Thread-aware memory access scheduling policies utilize the information that obtains from the memory access behavior of each thread. Parallelism-aware batch scheduling (PAR-BS) uses request batching to ensure fairness of service to threads sharing a memory channel [20]. The memory controller groups a fixed number of the oldest requests from each thread into a batch and services all the requests in the current batch before forming a new batch (Figure 2.11). This request batching provides fairness by alleviating starvation of requests and threads. It also allows parallelism of each thread to be optimized within a batch. To maximize throughput of batch processing, the memory controller exploits row-buffer locality and intra-thread bank parallelism in a

batch. In a batch, requests with high row-buffer locality are prioritized over requests with low row-buffer locality, and requests from threads with high ranks are prioritized over requests from low ranks. Thread ranking follows the shortest job first principle that gives the non-memory intensive threads higher priority than memory intensive threads.

Adaptive per-thread least-attained-service (ATLAS) [27] is a scheme that allows multiple memory controllers to coordinate their scheduling decision to improve throughput. The execution time is split into multiple epochs. During each epoch, the memory controllers profile the amount of service that each thread received. At the beginning of the next epoch, this profiled information is gathered at a central coordinator, which increases the priorities of the threads that received the least service in the previous epoch. This information is propagated to the memory controllers. Therefore, the threads attained the least service are prioritized over other threads in the new epoch. Thread cluster memory scheduler (TCM) [26] uses the memory access behavior of the thread to decide its priority. First, it prioritizes requests from non-memory-intensive threads over memory-intensive ones during memory scheduling. After making the observation on unfairness caused by interference among memory-intensive threads, TCM periodically shuffles the priority order among such threads to increase fairness. However, regardless of the shuffling, threads with higher bank-level parallelism are prioritized over streaming threads that have high row-buffer locality.

Ipek et al. [21] and Mukudan et al. [22] pointed out that above-mentioned scheduling policies have limited abilities to adapt themselves to the dynamic change of memory access behavior. To address this problem, they proposed reinforcement learning-based (RL-based) memory controllers which learn optimal control policies through interaction with their system environment. The RL-based memory controllers keep track of their system state to schedule DRAM commands that are estimated to give the highest long-term reward in the current state. In order to adapt changes in the memory access behavior of workload, the controllers continuously update reward values assigned to state-action pairs with feedbacks from the systems.

2.3.1 Page Management Policies

Page-management policies are generally a part of scheduling policies. Two static policies (open and close) are widely used. When there is no request to be serviced in recently accessed DRAM page, the open-page policy keeps this page activated (open), but the close-page policy deactivates this page. The static policies are easy to implement, but performance varies depending on memory access patterns of applications. As shown in Table 2.2, applications show different row-buffer conflict rate. The row-buffer conflict rate of 429.mcf is 84.6%, but that of 462.libquantum is 19.3%.

Application	RB Conflict	Application	RB Conflict
400.perlbench	37.7%	453.povray	8.3%
401.bzip2	49.0%	454.calculix	12.1%
403.gcc	40.7%	456.hmmer	59.7%
410.bwaves	47.7%	458.sjeng	50.2%
416.gamess	50.7%	459.GemsFDTD	80.0%
429.mcf	84.6%	462.libquantum	19.3%
433.milc	22.9%	464.h264ref	39.5%
434.zeusmp	50.0%	465.tonto	5.9%
435.gromacs	34.3%	470.lbm	37.4%
436.cactusADM	93.4%	471.omnetpp	52.3%
437.lestlie3d	45.5%	473.astar	55.8%
444.namd	5.7%	481.wrf	37.5%
445.gobmk	37.4%	482.sphinx3	26.2%
447.dealII	12.5%	483.xalancbmk	23.3%
450.soplex	44.2%		

Table 2.2: Row buffer (RB) conflict rate of individual applications in SPEC CPU2006 benchmark suite. Our detailed simulation setup is described in Section 5.4.

To address this problem, more sophisticated page-management policies have been proposed. Minimalist open-page policy [23] keeps the accessed DRAM page activated only for a predetermined time interval (tRC). For a read or a single-line prefetch request,

the memory controller does not deactivate the open page until tRC (~50ns) delay expires in order to service incoming row-buffer hit requests during the tRC window. However, the memory controller immediately deactivates the open page after processing multi-line prefetching. Intel Xeon 7560 supports close, open, and adaptive-open page-management policies, and one of these policies can be configured in BIOS [24]. Adaptive open-page policy uses per-bank idle timers. If a bank is idle for N clock cycles where N is parameterizable, the memory controller precharges the bank. Intel has implemented adaptive page-management (APM) technology [27] which tracks memory access patterns across open pages. The APM deactivates open pages based on their page access histories. The RL-based memory controllers dynamically deactivate an open page if precharge command gives highest long-term reward in the current state [21] [25].

In Section 5.3, we propose novel page management policy inspired by the idea of branch predictors. We evaluate the performance impacts of our proposed page-management policies over conventional policies in Section 5.4.4.

Chapter 3

Manycore Simulation Infrastructure

Multicore processors have already become mainstream. Emerging manycore processors have brought new challenges to the architecture research community, together with significant performance and energy advantages. Manycore processors are highly integrated complex system-on-chips (SoCs) with complicated core and uncore subsystems. The core subsystems can consist of a large number of traditional and asymmetric cores. For example, the Tilera Tile64 [4] has 64 small cores. The latest Intel Xeon Phi coprocessor [28] has more than 50 medium-size cores on a single chip. Moreover, ARM recently announced the first asymmetric multicore processor known as big.LITTLE [5], which includes a combination of out-of-order (OOO) Cortex-A15 (big) cores and in-order (IO) Cortex-A7 (little) cores. While the

---

This chapter is based on [2]. -

Cortex-A15 has higher performance, the Cortex-A7 is much more energy efficient. Using them at the same time, ARM big.LITTLE targets high performance and energy efficiency at the same time. The uncore subsystems of the emerging manycore processors have also become more powerful and complex than ever, with features such as larger and deeper cache hierarchies, advanced on-chip interconnects, and high performance memory controllers. For example, the Intel Xeon E7-8870 already has a 30MB L3 cache. Scalable Network-on-Chip (NoC) and cache coherency implementation efforts have also emerged in real industry designs, such as the Intel Xeon Phi [28]. Moreover, emerging manycore designs usually require system software (such as OSes) to be heavily modified or specially patched. For example, current OSes do not support the multi-processing (MP) mode in ARM big.LITTLE, where both fat A15 cores and thin A7 cores are active. A special software switcher [5] is needed to support thread migration on the big.LITTLE processor.

Simulators have been prevalent tools in the computer architecture research community to validate innovative ideas, as prototyping requires significant investments in both time and money. Many simulators have been developed to solve different research challenges, serving their own purposes. However, new challenges brought by emerging (asymmetric) manycore processors as mentioned above demand new simulators for the research community. As discussed in the simulator taxonomy analysis in Section 3.1, while high-level abstraction simulators are not

appropriate for conducting microarchitectural research on manycore processors, full-system simulators usually are relatively slow, especially when system/OS events are not the research focus. Moreover, with unsupported features in existing OSes, such as the asymmetric ARM big.LITTLE processor, larger burdens are placed on researchers, especially when using a full-system simulator. Thus, a *lightweight*, *flexible*, and *detailed microarchitecture-level* simulator is necessary for research on emerging manycore microarchitectures.

We categorize widely used simulators in academia and explain requirements to simulate manycore system as an applications-level+ simulator in Section 3.1. In Section 3.2, we describe the overall simulation framework of McSimA+ by explaining interaction between the functional simulator (front-end) and the timing simulator (back-end). We also explain the detailed organization of the thread management layer of McSimA+. We show the organization of the timing simulator which models manycore architecture in Section 3.3. We evaluated the accuracy of McSimA+ in Section 3.4. As a case study, we propose a novel asymmetric manycore architecture and evaluate it in Section 3.5. Finally, we present the limitations and scope of McSimA+ in Section 3.6.

Simulators	FS/A	DC	μAr	X86	Mc	SS
gem5 [29]	FS	N	Y	Y	P*	+
GEMS [30]	FS	Y	Y	N†	P*	+
MARSSx86 [31]	FS	Y	Y	Y	P*	+
SimFlex [32]	FS	Y	Y	N†	P*	+
PTLsim [33]	FS	Y	Y	Y	P*	+
Graphite [34]	A	Y	N	Y	Y	+++
SESC [35]	A	N	Y	N†	N	++
Sniper [36]	A	Y	N	Y	Y	+++
SimpleScalar [37]	A	N	N	N†	N	++
Booksim [38]	N/A	N/A	N/A	N/A	N	++
Garnet [39]	N/A	N/A	N/A	N/A	N	++
GPGPUsim [40]	A	Y	Y	N/A	N	++
DRAMSim [41]	N/A	N/A	N/A	N/A	N	++
Dinero IV [42]	A	N	N	N/A	N	++
Zesto [43]	A	N	N	Y	N	+
CMP$im [44] [45]	A	Y	Y	Y	N	++
Preferred	A+	Y	Y	Y	Y	≥++

Table 3.1: Summary of existing simulators categorized by features. Abbreviations (details in main text): (FS/A)−Full−system (FS) vs. Application−level (A), (DC)−Decoupled functional and perfor−mance simulations, (μAr)−Microarchitecture details, (x86)−x86 ISA support, (Mc)−Manycore support, (SS)−Simulation speed; (A+)−A middle ground between full−system and application−level simulation, (Y)−Yes, (N)−No, (N/A)−Not applicable, (P)−Partially supported, †x86 is not fully supported for manycore. *Manycore (e.g. 1,000 cores and beyond) is not fully supported due to emulators/host OSes. A preferred manycore simulator should be lightweight (A+ and DC) and reasonably fast, without support of Mc, μAr, and x86. Unlike other simulators, the CMP$im family is not publicly available.

3.1 Why Yet Another Simulator?

Numerous processor and system simulators are already available as shown in Table 3.1. All of these simulators have their own merits and serve their different purposes well. McSimA+ was developed to enable detailed asymmetric manycore microarchitecture research.

For a better understanding of the need of another simulator for the abovementioned purpose, we explore the space of the existing simulators and explain why those do not cover the study we want to conduct. Table 3.1 shows the taxonomy of the existing simulators with the following six dimensions: 1) full-system vs. application-level simulation (FS/A), 2) decoupled vs. integrated functional and performance simulation (DC), 3) microarchitecture-level (i.e., cycle-level) vs. high-level abstract simulation (μAr), 4) supporting x86 or not, 5) whole manycore system support or not (Mc), and 6) the simulation speed (SS).

Full-system (FS) vs. application-level simulation (A)

Full-system simulators, such as gem5 [29] (full-system mode), GEMS [30], MARSSx86 [31], and SimFlex [32] run both applications and system software (mostly OSes). A full-system simulator is particularly beneficial when the simulation involves heavy I/O activities or extensive OS kernel function support. However, these simulators are relatively slow and make it difficult to isolate the impact of architectural changes from the interaction between hardware and software stacks. Moreover, because they

rely on existing OSes, they usually do not support manycore simulations well. They also typically require research on both the simulator and the system software at the same time, even if the research targets only architectural aspects. For example, current OSes (especially Linux) do not support manycore processors with different core types; thus, OSes must be changed to support this feature. In contrast, these aspects are the specialties of application-level simulators, such as SimpleScalar [37], gem5 [29] (system-call emulation mode), SESC [35], and Graphite [34] along with its derivative Sniper [36]. However, a pure application-level simulation is insufficient, even if I/O activity and time/space sharing are not the main areas of focus. For example, thread scheduling in a manycore processor is important for both performance accuracy and research interests. Thus, it is desirable for application-level simulators to manage threads independently from the host OS and the real hardware on which the simulators run.

Decoupled vs. integrated functional and performance simulation (DC)

Simulators need to maintain both functional correctness and performance accuracy. Simulators such as gem5 [29] choose a complex "execute-in-execute" approach that integrates functional and performance simulations to model microarchitecture details with very high levels of accuracy. However, to simplify the development of the simulator, some simulators trade modeling details and accuracy for reduced complexity and decouple functional simulation from performance simulation by offloading the functional

simulation to third party software, such as emulators or dynamic instrumentation tools, while focusing on evaluating the performance of new architectures with benchmarks. This is acceptable for most manycore architecture studies, where reasonably detailed microarchitecture modeling is sufficient. For example, GEMS [30] and SimFlex [32] offload functional simulations to Simics [46], PTLSim [33] and its derivative MARSSx86 [31] offloads functional simulations to QEMU [47], and Graphite [34] and its derivative Sniper [36] offload functional simulations to Pin [48].

Details (μAr) vs. simulation speed (SS)

A manycore processor is a highly integrated complex system with a large number of cores and complicated core and uncore subsystems, leading to a tradeoff between simulation accuracy and speed. In general, the more detailed an architecture the simulator can handle, the slower the simulator simulation speed. For example, Graphite [34] uses less detailed models, such as the one-IPC model, to achieve better simulation speed. Sniper [36] uses better abstraction methods such as interval-based simulation to gain more accuracy with less performance overhead. While these simulators are good for early stage design space explorations, they are not sufficiently accurate for detailed microarchitecture-level studies of manycore architectures. Graphite [34] and Sniper [36] are considered faster simulators because they use parallel simulation to improve the simulation speed. Trace-driven simulations can also be used to trade simulation accuracy for speed. However, these are not

suitable for multithreaded applications because the real-time synchronization information is usually lost when using traces. Thus, execution-driven simulations (i.e., simulation through actual application execution) are preferred. On the other hand, full-system simulators model both microarchitecture-level details and OSes. Thus, they sacrifice simulation speed for accuracy. Zesto [43] focuses on very detailed core-level microarchitecture simulations, which results in even lower simulation speeds. Instead, it is desirable to have a simulator to model manycore micro-architecture details while remaining faster than full-system simulators, which have both hardware and software overhead.

Support of manycore architecture (Mc)

As this paper is about simulators for emerging (asymmetric) manycore architectures, it is important to assess existing simulators on their support of (asymmetric) manycore architectures. Many simulators were designed with an emphasis on one subsystem of a manycore system. For example, Booksim [38] and Garnet [39] focus on NoC; Dinero IV [42] and CMP$im [44] [45] family focus on the cache; DRAMsim [41] focuses on the DRAM main memory system, Zesto [43] focuses on cores with limited multicore support, and GPGPUSim [40] focuses on GPUs. Full-system simulators support multicore simulations but require non-trivial changes (especially to the OS) to support manycore systems stably with a large number (e.g., more than 1,000) of asymmetric cores. Graphite [34] and Sniper [36] support manycore systems but lack

microarchitecture-level details, as mentioned earlier.

Support of x86 (x86)

While it is arguable as to whether an ISA is a key feature for simulators given that many researches do not need support for a specific ISA, supporting the x86 ISA has advantages in reality because most studies are done on x86 machines. For example, complicated cross-platform tool chains are not needed in a simulator with x86 ISA support. As shown in Table I, while existing simulators serve their purposes well, research on emerging (asymmetric) manycore processors prefers a new simulator that can *accurately* model the *microarchitecture* details of *manycore* systems. The new simulator is better at avoiding the weight of modeling both hardware and OSes so as to be *lightweight* yet still capable of controlling *thread management* for manycore processors. McSimA+ was developed specifically to fill this gap.

3.2 McSimA+: Overview and Operation

McSimA+ is a cycle-level detailed microarchitecture simulator for multicore and emerging manycore processors. Moreover, McSimA+ offers full control over thread/process management for manycore architectures, so it represents a middle ground between a full-system simulator and an application-level simulator. We refer to this as an *application-level+* simulator henceforth.

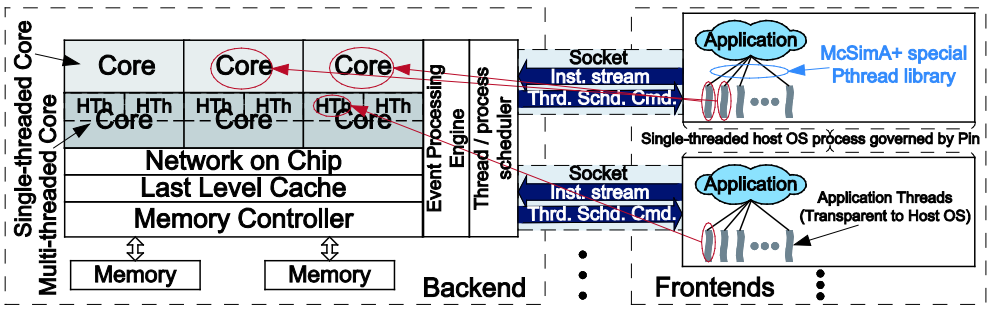


Figure 3.1: McSimA+ infrastructure. Abbreviations: "Inst. stream"–instruction stream, "Thrd. Schd. Cmd"–thread scheduling commands.

McSimA+ enjoys the light weight of an application-level simulator and better control of a full-system simulator. Moreover, its thread management layer makes implementing new functional features in emerging manycore processors much easier than changing the OSes with full-system simulators. McSimA+ supports detailed microarchitecture-level modeling not only of the cores, such as OOO, in-order, multi-threaded, and single-threaded cores, but also of all uncore components, including caches, NoCs, cache-coherence hardware, memory controllers, and main memory.

Moreover, innovative architecture designs such as asymmetric manycore architectures and 3D stacked main-memory systems are also supported. By supporting the microarchitectural details and rich features of the core and uncore components, McSimA+ facilitates holistic architecture research on multicore and emerging manycore processors. McSimA+ is a simulator capable of decoupled functional simulations and timing simulations. As shown in Figure 3.1, there are two main areas in the infrastructure of

McSimA+: 1) the Pin [48] based frontend simulator (*frontend*) for functional simulations and 2) the event-driven backend simulator (*backend*) for timing simulations.

Each frontend performs a functional simulation of a multithreaded workload using dynamic binary instrumentation using Pin and generates the instruction stream for the backend timing simulation. Pin is a dynamic instrumentation framework that can instrument an application in the granularity of an instruction, a basic block, or a function. Applications being executed are instrumented by Pin and the information of each instruction, function call, and system call is delivered to the McSimA+ frontend. After being processed by the frontend, the information is delivered to the McSimA+ backend, where the detailed target system including cores, caches, directories, on-chip networks, memory controllers, and main-memory subsystems are modeled. Once the proper actions are performed by the components affected by the instruction, the next instruction of the benchmark is instrumented by Pin and sent to the backend via the frontend. The frontend functional simulator also supports *fast forward*, an important feature necessary to skip instructions until the execution reaches the simulation region of interest.

The backend is an event-driven component that improves the performance of the simulation. Every architecture operation (such as a TLB/cache access, an instruction scheduling, and an NoC packet traversal) triggered by instruction processing generates a unique event with a component-type attribute (such as a core, a

cache, and an NoC) and a time stamp. These events are queued and processed in a global event-processing engine. When processing the events, a series of architecture events may be induced in a chain reaction manner; the global processing engine shown in Figure 2.1 processes all of the events in a strict timing order. If events occur in a single cycle, the simulation is performed in a manner similar to that of a cycle-by-cycle simulation. However, if no event occurs in a cycle, the simulator can skip the cycle without losing any information. Thus, McSimA+ substantially improves the simulation speed without a loss of cycle-level accuracy compared to cycle-driven simulators.

3.2.1 Thread Management for Application-level+ Simulation

Although McSimA+ is not a full-system simulator, it is not a pure application-level simulator either. Given that a manycore processor includes a large number of cores, hardware threads, and complicated uncore subsystems, a sophisticated thread/process management scheme is needed. OSes and system software usually lag behind the new features in emerging manycore processors; thus, modifying OSes for fullsystem simulators is a heavy burden. Therefore, it is important to gain full control of thread management for manycore microarchitecture-level studies without the considerable overhead of a full-system simulation. By using thread management layer and by taking full control over thread management from the host OS, McSimA+ is an application-level+

simulator that represents a middle ground between a full-system simulator and an application-level simulator.

The fact that it is an application-level+ simulator is also important in how it reduces simulation overhead and improves performance accuracy. As a decoupled simulator, McSimA+ leverages Pin by executing applications on native hardware to achieve a fast simulation speed. One way to support a multithreaded application in this framework is to let the host OS (we borrow the terms used on virtual machines) orchestrate the control flow of the application. However, this approach has two drawbacks. First, it is difficult to micro-manage the execution order of each thread governed by the host OS.

The timing simulator can make progress only if all the simulated threads held by all cores receive instructions to be executed or are explicitly blocked by synchronization primitives, whereas the host OS schedules the threads based on its own policy without considering the status of the timing simulator. This mismatch requires huge buffers to hold pending instructions, which is especially problematic for manycore simulations [49]. Second, if an application is not race free, we must halt the progress of a certain thread if it may change the flow of other threads that are pending in the host OS but may also be executed at an earlier time on the target architecture simulated in the timing simulator, which is a very challenging task.

3.2.2 Implementing the Thread Management Layer

When implementing the thread management layer in McSimA+ for an application-level+ simulation, we leveraged the solution proposed by Pan et al. [50] and designed a special Pthread [51] library implemented as part of the McSimA+ frontend. This Pthread library enables McSimA+ to manage threads completely independently of the host OS and the real system according to the architecture status and characteristics of the simulated target manycore processors. There are two major components in the special Pthread library: the Pthread controller and the Pthread scheduler. The Pthread controller handles all Pthread functionalities, such as pthread create, pthread destroy, pthread mutex, pthread local storage and stack management, and thread-safe memory allocation.

The thread scheduler in our special Pthread library is responsible for blocking and resuming threads during thread join, mutex/lock competition, and conditional wait operations. Existing Pthread applications can run on McSimA+ without any change of the code. An architect only needs to link to the special Pthread library rather than to the native one. During execution, all Pthread calls are intercepted by the McSimA+ frontend and replaced with the special Pthread calls. In order to separate thread execution from the OS, a multithreaded application appears to be a single threaded process from the perspective of the host OS/Pin. Thus, OS/Pin is not aware of the threads in the host OS process and surrenders the

full control of thread management and scheduling to McSimA+. In order to simulate unmodified multi-programmed workloads (each workload can be a multithreaded application), multiple frontends are used together with a single backend timing simulator. All frontends are connected to the backend via inter-process communication (sockets). All threads from the frontend processes are mapped to the hardware threads in the backend and are managed by the process/thread scheduler in the backend (Figure 3.1).

The thread scheduler in the Pthread library in the frontend maintains a queue of threads and schedules a particular thread to run when the backend needs the instruction stream from it. We implemented a global process/thread scheduler in the backend that controls the execution of all hardware threads on the target manycore processor. While the frontend thread scheduler manages threads according to the program information (i.e., the thread function calls), the backend process/thread scheduler has the global information (e.g. cache misses, resource conflicts, branch mispredictions, and other architecture events) of all of the threads in all processes and manages all of the threads accordingly.

The backend scheduler sends the controlling information to the individual frontends to guide the thread scheduling process in each multithreaded application, with the help of the thread scheduler in the special Pthread libraries in the frontends. Different thread scheduling policies (the default is round-robin) can be implemented to study the effects of scheduling policies on the simulated system.

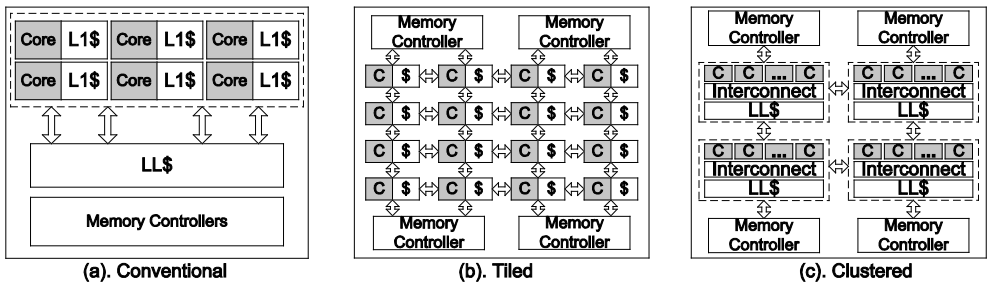


Figure 3.2: Example manycore architectures modeled in McSimA+. (a) shows a fully connected (with a bus/crossbar) multicore processor such as the Intel Nehalem [52] and Sun Niagara [53] processors, where all cores directly share all last-level caches through the on-chip fully connected fabric. (b) shows a tiled architecture, such as the Tilera Tile64 [4] and Intel Knights Corner [28], where cores and local caches are organized as tiles and connected through a ring or a 2D-mesh NoC. (c) shows a clustered manycore architecture as proposed in [5], [54], [55], where on-chip core tiles first use local interconnects to form clusters that are then connected via ring or 2D-mesh NoC.

3.3 Modeling of Manycore Architecture

The key focus of McSimA+ is to provide fast and detailed microarchitecture simulations for manycore processors. McSimA+ also supports flexible manycore designs. Figure 3.2 shows a few examples of the flexibility of McSimA+ in modeling different manycore architectures from a fully connected multicore processor (Figure 3.2(a)), such as the Intel Nehalem [52] and Sun Niagara

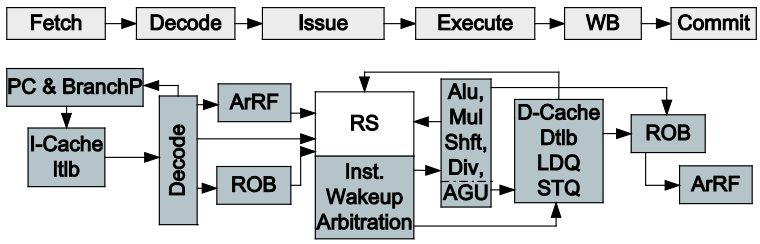
[53], to tiled architectures (Figure 3.2(b)), such as the Tilera Tile64 [4] and Intel Knights Corner [28], and to clustered manycore architectures (Figure 3.2(c)) as in ARM big.LITTLE [5]. Moreover, McSimA+ supports a wide spectrum of innovative and/or emerging technologies, such as asymmetric cores [5] and 3D main memory [56]. By supporting detailed and flexible manycore architecture modeling, McSimA+ facilitates comprehensive and holistic research on multicore and manycore processors.

3.3.1 Modeling of Core Subsystem

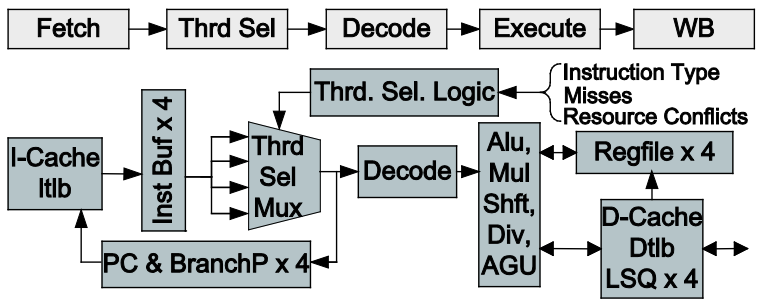
McSimA+ supports detailed and realistic models of the scheduling units based on existing processor core designs, including in-order, OOO, and multithreaded core architectures. Figure 3.3 shows the overall core models in McSimA+ for OOO and in-order cores. We depict the cores as a series of units and avoid calling them "pipeline stages," as they are high-level abstractions of the actual models in McSimA+ and because many detailed models of hardware structures (e.g., L1 caches and reservation stations) are implemented within these generic units.

Modeling of Out-of-Order Cores

The OOO core architecture in McSimA+ has multiple units, including the fetch, decode, issue, execution (exec), write-back, and commit stages, as shown in Figure 3.3(a).



(a). OOO Superscalar Core Model



(b). In-order Core (w. interleaved multithreading) Model

Figure 3.3: Core modeling in McSimA+.

The fetch unit reads a cache line containing multiple instructions and stores the instructions in an instruction stream buffer. By modeling the instruction stream buffer, McSimA+ ensures that the fetch unit only accesses the TLB and instruction cache once for each cache line (with multiple instructions) rather than for each instruction. As pointed out in earlier work [43], most other academic simulators fail to model the instruction stream buffer and generate a separate L1-I$ request and TLB request for each instruction, which leads to overinflated accesses to the L1-I$ and TLB and subsequent incorrect simulation results. Next, instructions are taken from the instruction stream buffer and

decoded. Because McSimA+ obtains its instruction stream from the Pin-based frontend, it can easily assign different latency levels based on the different instruction types and opcodes. The issue unit assigns hardware resources to the individual instructions. By default, McSimA+ models the reservation-station (RS)-based (data-capture scheduler) OOO core following the Intel Nehalem [52]/P6 [57] microarchitectures.

McSimA+ allocates a reorder buffer (ROB) entry and an RS entry to each instruction. If either resource is full, the instruction issue stalls until both the ROB and RS have available entries. Once instructions are issued to the RS, the operands available in either the registers or the ROB are sent to the RS entry. The designators of the unavailable source registers are also copied into the RS entry and are used for matching the results from functional units and waking up proper instructions; thus, only true read-after-write data dependencies may exist among the instructions in the RS.

The execution unit handles the dynamic scheduling of instructions, their movement between the reservation stations and the execution units, the actual execution, and memory instruction scheduling. While staying in the RS, instructions wait for their source operands to become available so that they can be dispatched to execution units. If the execution units are not available, McSimA+ does not dispatch the instructions to execute, even if the source operands of the instructions are ready. It is possible for multiple instructions to become ready in the same cycle. McSimA+ models the bandwidth of each execution unit, including both integer

ALUs, floating point units, and load/store units. Instructions with operands ready bid on these dispatch resources, and McSimA+ arbitrates and selects instructions based on their time stamps to execute on the proper units. Instructions that fail in the competition have to stall and try again at the next cycle. For load and store units, McSimA+ assumes separate address generation units (AGU) are available for computing addresses as in the Intel Nehalem [52] processor.

The write-back unit deals with writing back the results of both non-memory and memory instructions. Once the result is available, McSimA+ will update both the destination entry in the ROB and all entries with pending results in the RS. The RS entry will be released and marked as available for the next instruction. The commit unit completes the instructions, makes the results globally visible to the architecture state, and releases hardware resources. McSimA+ allows the user to specify the commit width.

Modeling of In-Order Cores

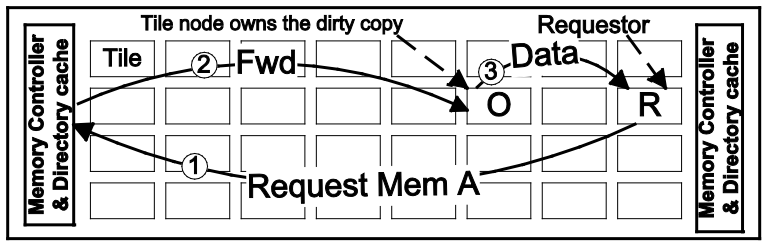
Figure 3.3(b) shows an in-order core with fine-grained interleaved multithreading as modeled in McSimA+. The core has six units, including the fetch, decode, select, execution (exec), memory, and write-back units. For an in-order core, the models of the fetch and decode units are similar to those of OOO cores, while the models of execution and writeback units are much simpler than those for OOO cores. For example, the model of the instruction scheduling structure for in-order cores in McSimA+ degenerates

to a simple instruction queue. Figure 3.3(b) also shows the modeling of interleaved multithreading in McSimA+. This core model closely resembles the Sun Niagara [53] processor.

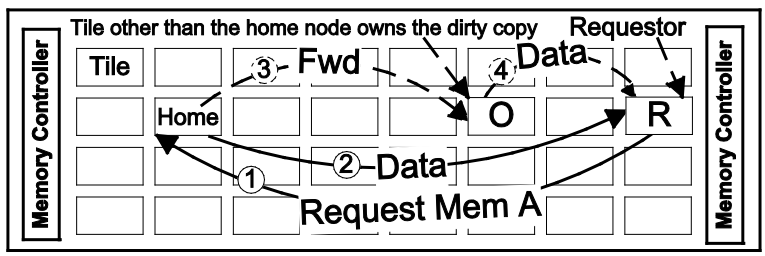
McSimA+ models the thread selection unit after the fetch unit. McSimA+ maintains the detailed status of each hardware thread and selects one to execute on the core pipeline every cycle in a round-robin fashion from all active threads. A thread may be removed from the active list for various reasons. Threads can be blocked and marked as inactive by the McSimA+ backend due to operations with a long latency, such as cache misses and branch mispredictions or by the McSimA+ frontend thread scheduler owing to the locks and barriers within a multithreaded application. When selecting the thread to run in the next cycle, McSimA+ also considers resource conflicts such as competitions pertaining to execution units. McSimA+ arbitrates the competing active threads in a round-robin fashion, and a thread that fails will wait until the next cycle.

3.3.2 Modeling of Cache and Coherence Hardware

McSimA+ supports highly detailed models of cache hierarchies (such as private, coherent, shared, and non-blocking caches) to provide detailed microarchitecture-level modeling for both core and uncore subsystems in manycore processors. Faithfully modeling coherence protocol options for manycore processors is critical to model all types of cache hierarchies correctly.



(a) DRAM-dir



(b) Duplicate-tag and Sparse-dir, both with home nodes

Figure 3.4: Cache coherence microarchitecture modeling in McSimA+. In DRAM-dir (a) model, each tile contains core(s), private cache(s), local interconnect (if necessary) within a tile, and global interconnect for inter-tile communications. Directory caches are co-located with memory controllers. In Duplicate-tag and Sparse-dir (b), McSimA+ assumes that directory is distributed across the tiles using the *home node* concept [28], [53], [58]. Thus, the tiles in (b) have the extra directory information although not shown in the figure.

Because McSimA+ supports flexible compositions of cache hierarchies, the last-level cache (LLC) can be either private or shared. The address-interleaved shared LLC has a unique location for each address, eliminating the need for a coherence mechanism. However, even when the LLC is shared, coherence between the upper-level private (e.g., L1 or L2) caches must be explicitly

maintained. Figure 3.4 shows the tiled architecture with a private LLC to demonstrate the coherence models in McSimA+. We assume directory-based coherence because McSimA+ targets future manycore processors that can have 64 or more cores, where frequent broadcasts are slow, difficult to scale, and power-hungry.

McSimA+ supports three mainstream directory-based cache coherence implementations (to enable important tradeoff studies of the performance, energy, scalability, and complexity of different architectures): the DRAM directory with a directory cache (DRAM-dir, as shown in Figure 3.4(a)) as in the Alpha 21364 [59], the distributed duplicate tag (duplicate-tag, as shown in Figure 3.4(b)) as in the Niagara processors [53], [58], and the distributed sparse directory (sparse-dir, as shown in Figure 3.4(b)) [60].

DRAM-dir is the most straightforward implementation; it stores directory information in main memory with an additional bit-vector for every memory block to indicate the sharers. While the directory information is logically stored in DRAM, performance requirements may dictate it to be cached in the on-chip directory caches that are usually collocated at the on-chip memory controllers, as the directory cache has frequent interactions with main memory. Figure 3.4(a) demonstrates how the DRAM-dir is modeled in McSimA+. Each core is a potential sharer of a cache block. A cache miss triggers a request and sends it through the NoC to the appropriate memory controller based on address interleaving to where the target directory cache resides. The directory information is then retrieved. If the data is on chip, the directory information manages

the data forwarding between the owner and the sharers. If a directory cache miss/eviction occurs, McSimA+ generates memory accesses at a memory controller and fetches the directory information (and the data if needed) from the main memory.

McSimA+ supports both the duplicate-tag and the sparse-dir features to provide smaller storage overheads than DRAM-dir and to make the directory scalable for processors with a large number of cores. The duplicate-tag maintains a copy of the tags of every possible cache that can hold the block, and no explicit bit vector is needed for sharers. During a directory lookup operation, tag matches indicate finding by the sharers. The duplicate-tag eliminates the need to store and access the directory information in DRAM. A block not found in a duplicate tag is known to be uncached.

Despite its good coverage for all of the cached memory blocks, a duplicate-tag directory can be challenging as the number of cores increases because its associativity must equal the product of the cache associativity and the number of caches [61]. McSimA+ supports sparse-dir [62] as a low-cost alternative to the duplicate-tag directory. Sparse-dir reduces the degree of directory associativity but increases the number of directory sets. Because this operation loses the one-toone correspondence of directory entries to cache frames, each directory entry is extended with the bit vector for storing explicit sharer information. Unfortunately, the non-uniform distribution of entries across directory sets in this organization incurs set conflicts, forcing the invalidation of cached blocks tracked by the conflicting directory

entries and thus reducing the performance of the system. McSimA+ provides all these different designs to facilitate in-depth research of manycore processors.

As shown in Figure 3.4(a), a coherent miss in DRAM-dir generates NoC traffic, and the request needs to travel through the NoC to reach the directory even if the data is located nearby. In order to model scalable duplicate-tag directories and sparse-dirs, we model the home node-based distributed implementation as in the Intel Xeon Phi [28] and Niagara processors [53], [58], where the directory is distributed among all nodes by mapping a block address to the *home node*, as shown in Figure 3.4(b). We assume that home nodes are selected by address interleaving on low-order blocks or page addresses. A coherent miss first looks up the directory in the home node. If the home node has the directory and data, the data will be sent to the request directly via steps (1)-(2) shown in Figure 3.4(b). The home node may only have the directory information without the latest data, in which case the request will be forwarded to the owner of the copy and the data will be sent from there via steps (1), (3), and (4), as shown in Figure 3.4(b). If a request reaches the home node but fails to find a matching directory entry, it allocates a new entry and obtains the data from memory. The retrieved data is placed in the home tile's cache and a copy is returned to the requesting core. Before victimizing a cache block with an active directory state, the protocol must first invalidate sharers and write back dirty copies to memory.

3.3.3 Modeling of Network-on-Chips (NoCs)

McSimA+ supports different on-chip interconnects, including buses, crossbars, and multi-hop NoCs with various topologies, including ring and 2D mesh topologies. A multihop NoC has links and routers, where the per-hop latency is a tunable parameter. As shown in Figure 3.2, McSimA+ supports a wide range of hierarchical NoC designs, where cores are grouped into local clusters and the clusters are connected by global networks.

The global interconnects can be composed of buses, crossbars, or multi-hop NoCs. McSimA+ models different message types (e.g., data blocks, addresses, and acknowledgements) that route in the NoC of a manycore processor. Multiple protocol-level virtual channels in the NoC are used to avoid deadlocks in the on-chip transaction protocols. A protocol-level virtual channel is also modeled to have multiple virtual channels inside to avoid a deadlock within the NoC hardware and improve the performance of the network.

McSimA+'s detailed message and virtual channel models not only guarantee simulation correctness and performance accuracy but also facilitate important microarchitecture research on NoCs. For example, when designing a manycore processor with a NoC, it is often desirable to have multiple independent logical networks for deadlock avoidance, privilege isolation, independent flow control, and traffic prioritization purposes. However, it is an interesting design choice as to whether the different networks should be

implemented as logical or virtual channels over one large network, as in the Alpha21364 [59], or as independent physical networks as in Intel Xeon Phi [28]. An architect can conduct in-depth studies of these alternatives using McSimA+.

3.3.4 Modeling of the Memory Controller and Main Memory

McSimA+ supports detailed modeling of memory controllers and main-memory systems. First, the placement of memory controllers, an important design choice [63], can be freely determined by the architects. As shown in Figure 3.2, the memory controllers can be connected by crossbars/buses and placed at edges. They can also be distributed throughout the chip and connected to the routers in the NoC. McSimA+ supports numerous memory scheduling policies, including FC-FRFS [11] and PAR-BS [20].

For each memory scheduling policy, an architect can further choose to use either open-page or close-page scheduling policies on top of the base scheduling policy. For example, if the PAR-BS policy is assumed to be the base memory scheduling policy, a close-page policy on top of it will close the DRAM page when there is no pending access in the scheduling queue to the current open DRAM page.

Moreover, the modeled memory controller also supports a DRAM power-down mode during which DRAM chips consume only a fraction of their normal static power but require extra cycles to enter and exit the state. When this option is chosen, the controller

will schedule the main memory to enter a power-down mode after the scheduling queue is empty and thus the attached memory system has been idle for a predefined interval. This facilitates research on trade-offs between power-saving benefits and performance penalties.

In order to model the main-memory system accurately, the main-memory timing is also rigorously modeled in McSimA+. For the current and near-future standard DDRx memory systems, McSimA+ includes user-adjustable memory timing parameters such as row activation latency, precharge latency, row access latency, column access latency, and the row cycle time with different banks.

3.4 Validation

There are two aspects in the validation of an execution-driven architectural simulator: functional correctness that guarantees programs to finish correctly and performance accuracy that ensures that the simulator faithfully reflects the performance of the execution, as if the applications were running on the actual target hardware. Functional correctness is typically straightforward to verify, especially for the simulators with decoupled functional simulations such as GEMS [30], SimFlex [32] and our McSimA+. We checked the correctness of the simulation results on SPLASH-2 using the correctness check option within each program.

Parameters	Values
Frequency (GHz)	2.53
Cores/chip	4
(ROB/RS) entry	128/36
(L1 I-TLB/L1 D-TLB) entry	128/64
(IF/CM/IS) width	4/4/6
L1 I-$	32KB, 4-way
L1 D-$	32KB, 8-way
L2$ per core	256KB, 8-way, inclusive
L3$ (shared)	8MB, 16-way, inclusive
Main memory	3 channels, DDR3-1333

Table 3.2: Configuration specifications of the validation target with Intel Xeon E5440 multi-core processor. IF/CM/IS stands for fetch/commit/issue.

However, performance accuracy is much more difficult to verify. Moreover, a recent trend (as in a recent workshop panel [64] with several industrial researchers) argues that provided that academic simulators can foster correct research insights through simulations the validation of the simulators against real systems is not necessary. This trend partially leads to the fact that the majority of existing academic simulators lack sufficient validation against real systems. However, we believe that a rigorous validation against actual hardware systems is required. We performed the validations in layers, first validating at the entire multicore processor level and then validating the core and uncore subsystems.

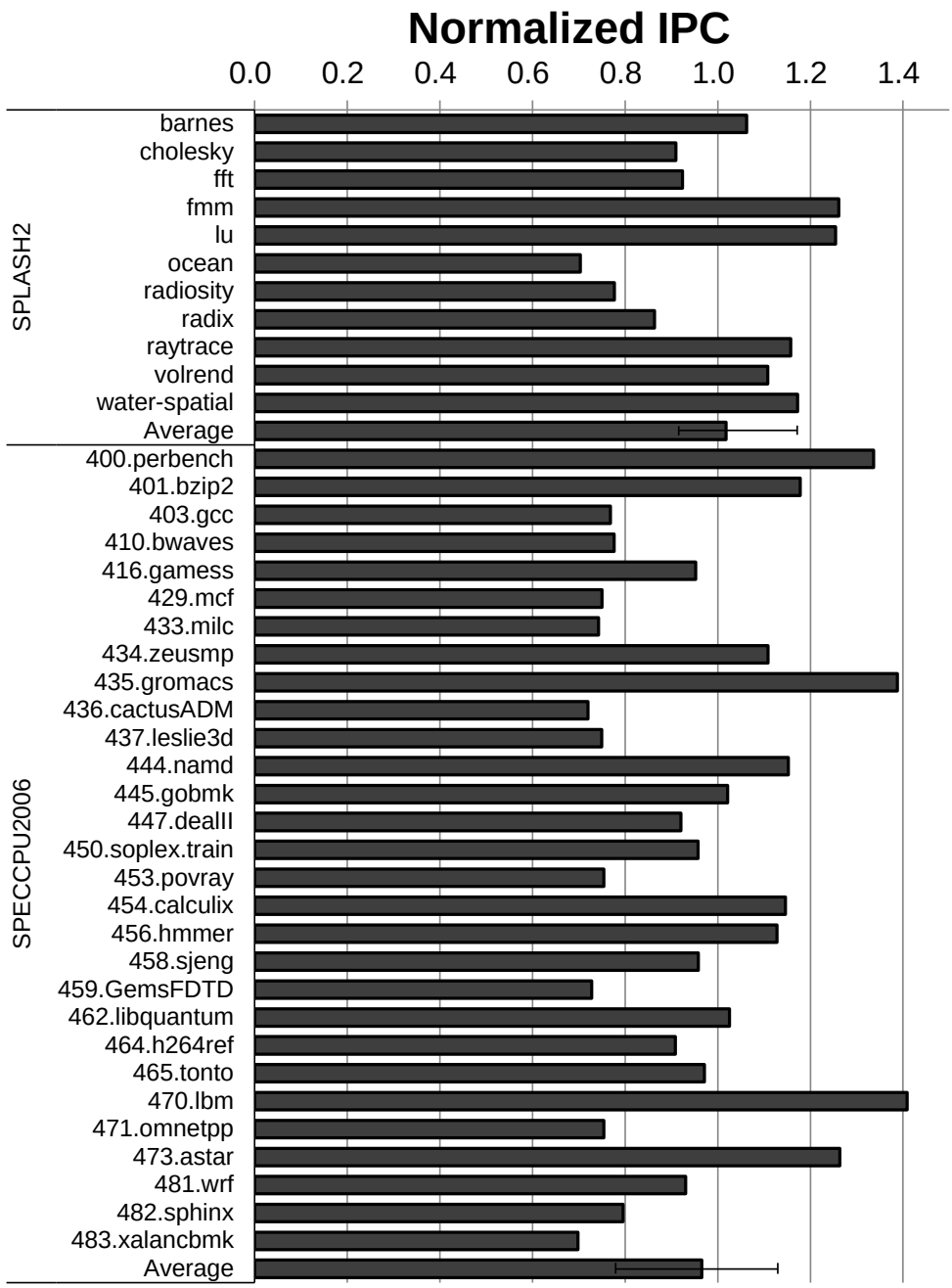


Figure 3.5: The relative IPC of McSimA+ simulation results normalized to that of native machines. We use the entire SPLASH-2 and SPEC CPU2006 benchmark suite.

The performance accuracy of McSimA+ at an overall multicore processor level was validated using the multithreaded benchmark suite SPLASH-2 [65] against an Intel Xeon E5540 (Nehalem [52]) based real server whose configuration specifications (listed in Table 3.2) were used to configure the simulated target system in McSimA+. For all of the validations, we turned off hyper-threading and the L2 cache prefetcher in the real server and configured McSimA+ accordingly. Figure 3.5 shows the IPC (Instructions Per Cycle) results of the SPLASH-2 simulations on McSimA+ normalized to the IPCs of the native executions on the real server as collected using Intel Vtune [66]. When running benchmarks on the real machines, we ran the applications multiple times to minimize the system noise. As shown in Figure 3.5, the IPC results of the SPLASH-2 simulations on McSimA+ are in good agreement with the native executions, which have an average error of only 2.1% (14.2% on average for absolute errors). Its standard deviation is also as low as 12%.

We then validated the performance accuracy of McSimA+ at the core level using SPEC CPU2006 benchmarks, which are good candidates for validation because they are popular and single-threaded. The same validation target shown in Table 3.2 was used. Figure 3.5 shows the IPC results of McSimA+ simulations normalized to native machine executions on the real server for SPEC CPU2006. The simulation results track the native execution result from the real server very well, with an average error of only 5.7% and a standard deviation of 17.7%.

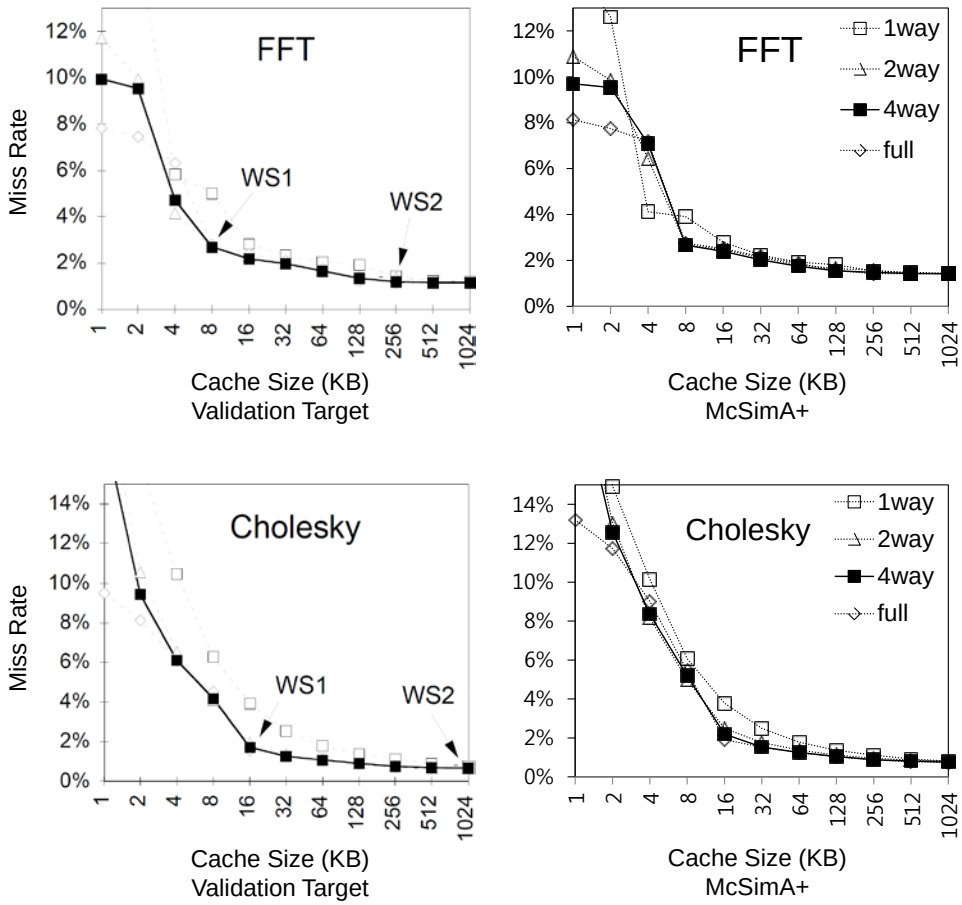


Figure 3.6: Validation of McSimA+ L2 cache simulation results to the simulation results from [65].

While the core subsystem validation is critical, the uncore subsystems of the processor are equally important. To validate the uncore subsystems, we focused on the last-level cache (LLC), as LLC statistics represent the synergy between cache/memory hierarchy and on-chip interconnects. We used SPLASH-2 benchmarks to validate the cache miss rates for the LLC, where both the cache size and the associativity vary to a large degree,

ranging from 1KB to 1MB and from one way to fully-associative, respectively. We used the results published in the original SPLASH-2 paper [65] as the validation targets because it is not practical to change the cache size or associativity on a real machine. We configured the simulated architecture as close as possible to the architecture (a 32-processor symmetric multiprocessing system) in the original paper [65].

Validation results on Cholesky and FFT are shown in Figure 3.6 as representatives. While FFT is highly scalable, Cholesky is dramatically different with poor scalability. As shown in Figure 3.6, the miss rate results obtained from McSimA+ very closely match the corresponding results reported in the earlier work [65]. For all SPLASH-2 benchmarks (including examples shown in Figure 3.6), the LLC miss rate difference between McSimA+ and the validation target does not exceed 2% over hundreds of data points collected at one time. This experiment demonstrates the high accuracy of McSimA+'s uncore subsystem models.

Our validation covers different processor configurations ranging from the entire multicore processor to the core and the uncore subsystems. The validation targets are comprehensive ranging from a real machine to published results. Thus, the validation stresses McSimA+ in a comprehensive and detailed way as well as tests its simulation accuracy with different processor architectures. In all validation experiments, McSimA+ demonstrates good performance accuracy.

Parameters	OOO (Nehalem [24]−like)	IO (Atom [67]−like
Issue width	6 (peak)	2
RS	36	N/A
ROB	128	N/A
LID cache	32KB, 8−way	16KB, 4−way
L2 cache	2MB 16−way	512KB, 16−way
Area (mm2)	6.56	2.15
Power (W)	3.97	0.66

Table 3.3: Parameters including area and power estimations from McPAT [54] of both OOO and IO cores.

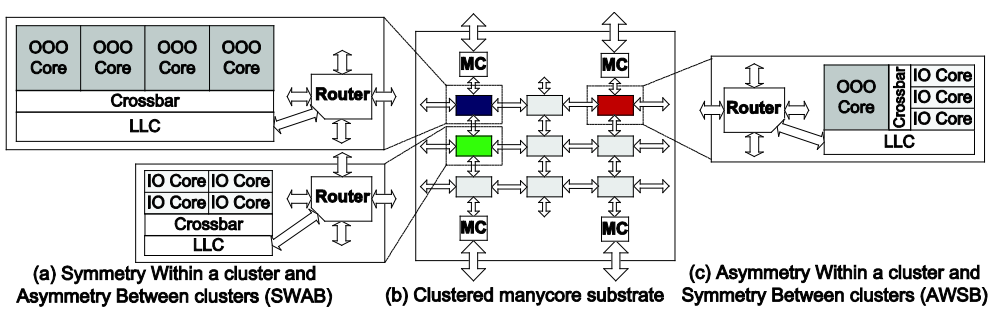


Figure 3.7: Clustered manycore architectures. (a) Symmetry Within a cluster and Asymmetry Between clusters (SWAB), fat OOO clusters (blue) and thin IO core (green). (b) Generic clustered manycore processor substrate. (c) Asymmetry Within a cluster and Symmetry Between clusters (AWSB) (red).

3.5 Clustering Effect in Asymmetric Manycore Processors

We illustrate the utility of McSimA+ by applying it to the study of clustering effects in emerging asymmetric manycore architectures. Asymmetric manycore processors, such as ARM big.LITTLE, have cores with different performance and power capabilities (e.g., *fat* OOO and *thin* in-order (IO) cores) on the same chip. Clustered manycore architectures (Figure 3.2(c)), as proposed in several studies [68], [54], [55] have demonstrated significant performance and power advantages over flat tiled manycore architectures (Figure 3.2(b)) due to the synergy of cache sharing and scalable hierarchical NoCs. Moreover, clustering has already been adopted in ARM big.LITTLE, the first asymmetric multicore design from industry. Despite the adoption of clustering in asymmetric multicore designs, effectively organizing clusters in a manycore processor remains an open question. Here, we perform a detailed study of clustered asymmetric manycore architectures to provide insights regarding this question.

3.5.1 Manycore with Asymmetric Within or Between Clusters

There are two clustering options for an asymmetric manycore design as shown in Figure 3.7. The first option is to have Symmetry Within a cluster and Asymmetry Between clusters (SWAB) as illustrated in Figure 3.7(a), where cores of the same type are

placed within a single cluster but where different clusters can have different core types. SWAB is the clustering option used in the ARM big.LITTLE design. The second option, which we propose, is to have Asymmetry Within a cluster and Symmetry Between clusters (AWSB), as illustrated in Figure 3.7(c). AWSB places different cores in a single cluster and forms an asymmetric cluster, but all clusters in a chip are symmetric despite the asymmetry within a single cluster.

Generally, thin (e.g., in-order) cores can achieve good performance for workloads with inherently high degrees of (static) instruction-level parallelism (ILP) (where ILP does not need to be dynamically extracted because the subsequent instructions in the stream are inherently independent), while fat (e.g., OOO) cores can easily provide good performance for workloads with hidden ILP (where the instructions in the stream need to be reordered dynamically to extract ILP). Thus, it is critical to run workloads on appropriate cores to maximize the performance gain and energy savings. In addition, the behavior of an application can vary at a fine-grained time scales during execution because of phase changes (e.g., a switch between computation-intensive and memory-intensive phases). Thus, frequent application/thread migrations may be necessary to fully exploit the performance and energy advantages of asymmetric manycore processors.

However, thread migrations are not free. In typical manycore architectures as shown in Figure 3.2, thread migrations cost, including the transfer of visible architecture states (e.g.,

transferring register files, warming up a branch prediction table and TLBs) and allowing invisible architecture states to become visible (drain a core pipeline, finish/abort speculation execution, for example); and 2) the cache data migration cost. In this paper, we focus on a heterogeneous multi-processing system (i.e., the MP mode of the big.LITTLE [5] processor), in which all cores are active at the same time.

Thus, a thread migration always involves at least a pair of threads/cores, and all cores involved in the migration will have new tasks to execute after the migration. The cache data migration cost varies significantly according to the cache architecture. Migration within a shared cache does not involve any extra cost, while migration among private caches requires the transfer of data from an old private cache to a new private cache. Although it can be handled nicely by coherence protocols without offchip memory traffic, data migration among private caches is still very expensive when the capacity of the last-level caches are large, especially when all cores involved in the thread migration will have new tasks to execute after the migration and thus will have to update their private caches.

Because the architecture-state migration cost is inevitable, it is critical to reduce the amount of cache data migration to support fine-grained thread migration so as to fully exploit the performance and energy advantages of asymmetric manycore processors. Thus, we propose AWSB, as in shown Figure 3.7(c) to support finer-grained thread migrations via its two-level thread migration

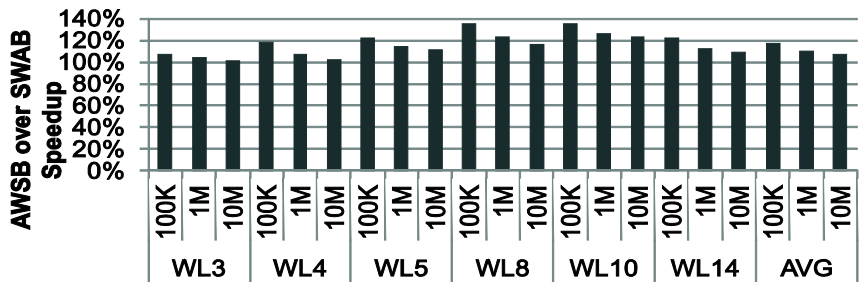
mechanism (i.e., intra-cluster and inter-cluster migrations). Because AWSB has clusters consisting of asymmetric cores, thread migration can be and is preferred within a cluster. Only when no candidates can be found within the same cluster (and the migration is very necessary to achieve higher performance and energy efficiency), an inter-cluster migration is performed. However, for SWAB, as shown in Figure 3.7(a), only high-overhead inter-cluster migrations are possible when the mapping between workloads and core types needs to be changed. Thus, by supporting two-level thread migrations, AWSB has the potential to reduce the migration cost and increase the migration frequency for a better use of the behavioral changes in the application execution and to achieve better system performance and energy efficiency than SWAB.

3.5.2 Evaluation

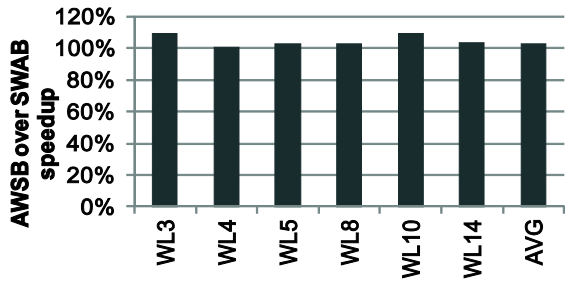
Using McSimA+, we evaluate our AWSB proposal, as shown in Figure 3.7(c), and compare it to the SWAB design adopted in the ARM big.LITTLE, as shown in Figure 3.7(a). We assume two core types (both 3GHz) are used in the asymmetric manycore processors, an OOO Nehalem [24]-like fat core and an in-order Atom [67]-like thin core. The parameters of both cores, including the area and the power estimations obtained from McPAT [54], are listed in Table 3.3.

SPEC CPU2006	445	458	400	453	483	471	473	470	437	410	459	450	429	436	482	433	464	465	403	401	416	447	462	444	434	454	456
WorkLoad \ Spdup	2.3	2.5	2.7	2.7	2.8	2.9	3.3	3.3	3.5	3.6	3.7	3.7	3.8	3.8	3.8	3.9	3.9	4.0	4.0	4.1	4.5	4.6	4.7	5.2	5.2	6.3	6.4
WL-1	2	3	3	5	2	1	1	1	3		1	3	1		1	1		3	1								
WL-2	2	3	3	2	2		3	2			1		1			2	1	5		1		1	2	1			
WL-3		2	1	1	1		2		2	1		1	1	2	1		1	2	2	3	2	2	1		2	2	
WL-4	1	1		1		1	1	1	1		2		1	1			3	1	1	1	4	2		2	2	1	4
WL-5										1	1	1		1		1		4	1	2	2	1	3	5	1	3	5
WL-6	2	1	5	1			2		2	1		2	2			2		3	2	2	3				1	1	
WL-7	1	3		4			1	1	3	2		2	2			1		2	3		3				1	2	1
WL-8			2		2		1	1	2		1	3	3		1	1	2	2	1	2	4		1	1	2		
WL-9	2	1	3	1	2			2	1	1	2	1	1	1		1	1	1		2			3			4	2
WL-10	3	1	1	1							1	2		1	1	3	2	3	3	2		3	1	1	2		1
WL-11	1	2	2	3		1	1		1		2	1	1		2	2	2		3	1	2	2				2	1
WL-12	2	3	2	1	1		3	2	2		3	2					1				2	2	2			1	3
WL-13	1	3	1	1	2		1	3	1	1	2	1				1	3	2	1	1	2	2			1	2	
WL-14	1	1	1		2		2	2	1	1	2	2	2		1	2	2	1	2		3	1		1		1	1
WL-15			1	1			1	2	1		2	2	1	2	1	1	1	1	3	1	3	1	1	1	3		2
WL-16	4	1	3		1		2		2		1	2	2	1	1	2	1	1	1	2	1	2	1			1	

Figure 3.8: Mixed workloads used in the case study constructed from SPEC CPU2006 benchmarks.



(a) Thread migration induced performance difference



(b) Speedup with optimized thread migration.

Figure 3.9: Performance comparison between SWAB and AWSB architectures.

We assume a core count ratio of fat cores to thin cores of 1:3 so that both fat and thin cores occupy a similar silicon area overall. Each fat core is assumed to have a 2MB L2 cache based on the Nehalem [24] design, while each thin core is assumed to have a 512KB L2 cache based on the Pineview Atom [67] design. Based on the McPAT [54] modeling results, a processor with 22nm technology with a ~260mm2 die area and a ~90W thermal design power (TDP) can accommodate 8 fat cores and 24 thin cores together with L2 caches, an NoC, and 4 single-channel memory controllers with DDR3-1600 DRAM connected.

The AWSB architecture has 8 clusters with each cluster containing 1 fat core and 3 thin cores. The SWAB architecture has 2 fat clusters each containing 4 identical fat cores and 6 thin clusters each containing 4 thin cores. All of the cores in a cluster share a multi-banked L2 cache via an intra-cluster crossbar. Because both AWSB and SWAB have 8 clusters, the same processor-level substrate as shown in Figure 3.7(b) is used with an 8-node 2D mesh NoC having a data width of 256 bits for inter-cluster communication. A two-level hierarchical directory-based MESI protocol is deployed to maintain cache coherency and to support private cache data migrations. Within a cluster, the L2 cache is inclusive and filters the coherency traffic between L1 caches and directories. Between clusters, coherence is maintained by directory caches associated with the on-chip memory controllers.

We constructed 16 mixed workloads, as shown in Figure 3.8

using the SPEC CPU2006 [69] suite for evaluating SWAB and AWSB. Because there are 32 cores on the chip in total, each of the workloads contains 32 SPEC CPU2006 benchmarks, and some benchmarks are used more than once in a workload. Some of the workloads (e.g., WL-5, as shown in Figure 3.8) contain more benchmarks with high IPC speedup, while others (e.g., WL-1) contain more benchmarks with low IPC speedup.

We first evaluated the thread migration overhead on the SWAB and AWSB architectures. We deployed all 32 benchmarks on all 32 cores for both SWAB and AWSB with the same benchmark to core mapping and then initiated a thread migration to change the mapping after an interval with 100K, 1M, or 10M instructions. The thread migration occurs during every interval until the simulation reaches 10 billion instructions or finishes earlier. Figure 3.9(a) shows the AWSB over SWAB speedup (measured as the ratio of the aggregated IPC) of the asymmetric 32 core processors.

As shown in Figure 3.9(a), AWSB demonstrated much higher performance, especially when the thread migration interval is small. For example, AWSB shows a 35% speedup over SWAB when running workload 8 (WL-8) at a thread migration interval of 100K instructions. On average, the AWSB architecture achieves 18%, 11%, and 8% speedup over the SWAB architecture with a thread migration interval of 100K instructions, 1M instructions, and 10M instructions, respectively. While the benchmark to core mapping changes from interval to interval, the SWAB and AWSB architectures have the same mapping at each interval. Thus, the

performance differences observed from Figure 3.9(a) are solely caused by the inherent differences in the thread migration overhead between the SWAB and AWSB architectures, and the results demonstrate AWSB' s better support of thread migration among the asymmetric cores.

We then evaluated the implications of the thread migration overhead on the overall system performance. We deployed 32 benchmarks in each workload to all cores in SWAB and AWSB with the same benchmark to core mapping scheme and then initiated a thread migration every 10M instructions. Unlike the previous study, in which SWAB and AWSB always have the same benchmark to core mapping so as to isolate the thread migration overhead, this study allows both SWAB and AWSB to select the appropriate migration targets for each benchmark. At the end of each interval, McSimA+ initiates a thread migration to place the high IPC speedup benchmarks on the fat cores with the low IPC speedup on the thin cores, as in earlier work [70].

As shown in Figure 3.9(b), AWSB demonstrates a noticeable performance improvement of more than 10% for workloads 3 and 8, with a 4% improvement on average for all 16 workloads. It is expected that the benefits of AWSB will be higher with finer-grained thread migrations, because the thread migration overhead of AWSB becomes much smaller than that of SWAB when moving to finer-grained thread migrations, as shown in Figure 3.9(a).

3.6 Limitations and Scope of McSimA+

There is no single "silver bullet" simulator that can satisfy all of the research requirements of the computer architecture community, and McSimA+ is no exception. Although it takes advantages of full-system simulators and application-level simulators by having an independent thread management layer, McSimA+ still lacks the support of system calls/codes (the inherent limitation of application-level simulators).

Therefore, research on OSes and applications with extensive system events (e.g. I/Os) is not suitable for McSimA+. Because the Pthread controller in the frontend Pthread library is specific to the thread interface, non-Pthread multithreaded applications cannot run on McSimA+ without re-targeting the thread interface despite the fact that the frontend Pthread scheduler and backend global process/thread scheduler are feasible despite the particular thread interface used. McSimA+ targets emerging manycore architectures with reasonably detailed microarchitecture modeling, and outside its scope it is most likely suboptimal as compared to other suitable simulators.

Another limitation is the modeling of speculative wrong-path executions. Because McSimA+ is a decoupled simulator that relies on Pin for its functional simulation, wrong-path instructions cannot be obtained naturally from Pin, as they were never committed in the native hardware and are thus invisible beyond the ISA interface. However, this limitation is different from the inherent limitation of

lacking the support of system calls. Although speculative wrong-path executions are not supported at this stage, they can be implemented via the context (architectural state) manipulation feature of Pin, as used to implement the thread management layer. The same approach can be employed to guide an application to execute a wrong path, roll back an architectural state, and execute a correct path.

Chapter 4

Energy-Efficient DRAM Array Organizations

DRAM has been widely used for main-memory storage for modern manycore systems. As process technology advances, several billion bits can be stored in a single DRAM chip or die [14].

Since the primary design goal of DRAM is to achieve high storage density and to lower manufacturing cost, it uses smaller transistors and narrower wires so that the bandwidth and especially latency of DRAM devices do not improve as fast as their storage capacity. This causes a performance bottleneck on manycore systems, which is called *memory walls*. Also the power consumption of DRAM chips is similar to or even surpasses the power consumption of cores on contemporary throughput-oriented systems so that they become energy bottlenecks as well [6].

This chapter is based on [3]. -

In order to overcome these limitations, modern DRAM architectures employ internal prefetch to achieve high bandwidth on sequential accesses and multiple banks for a better utilization on random accesses [11]. Recently, there have been proposals to modify main-memory architectures for better performance and energy efficiency either by reducing the number of DRAM chips or the regions in a chip to serve a memory request [6] [71], or by replacing copper-based global and off-chip interconnects into waveguides [72]. The advances in 3D die-stacking and Through-Silicon Via (TSV) technology can further lower the energy and latency to connect cores and main memory [15]. However, there has been little research on DRAM array organizations and their impacts on the performance and energy efficiency of manycore systems.

In this thesis, we explore the array organizations of main-memory DRAMs by varying the number of banks per DRAM rank and the page size of each bank. We modify CACTI [73], a widely used memory modeling tool, to evaluate the influences of varying the number of banks and the page size on the power, area, and timing of main-memory DRAMs. In order to assess the impacts of these array organizations on the system-level performance and energy efficiency, we simulate a chip-multiprocessor system with die-stacked main memory using multithreaded and multi-programmed workloads. CACTI results show that increasing the page size lowers the DRAM access time and improves area efficiency, but also increases the page activation energy. Cycle-

accurate simulation results show that the performance and energy efficiency of the tested chip-multiprocessor system are improved with more banks per DRAM rank, but there are sweet spots on the page size so that the system achieves the highest performance with 8KB pages and the best energy-delay product with 2KB pages.

We show the detailed DRAM array organizations in Section 4.1, and evaluate their system-level impacts in Section 4.2.

4.1 Energy-Efficient DRAM Array Organizations

Even though there is a large design space in organizing DRAM arrays, we focus on varying the number of banks per rank and the page size of each DRAM bank. In order to evaluate the power, area, and timing tradeoffs of the proposed DRAM array organizations, we modify the latest version of the CACTI memory modeling tool [73].

Figure 4.1 illustrates the layout of a manycore system we evaluate. DRAM dies are stacked on top of a manycore processor die, which is composed of multiple clusters connected through on-chip networks. Each cluster contains cores, caches, a router, and a memory controller. Each DRAM die is partitioned to multiple channels. This is similar to the Wide-IO DRAM [15], but here we place peripherals and off-chip I/Os at the center of each channel. Each channel operates independently, so it works as a DRAM rank in conventional memory modules. TSVs are used to connect a memory controller to multiple ranks, one per DRAM die.

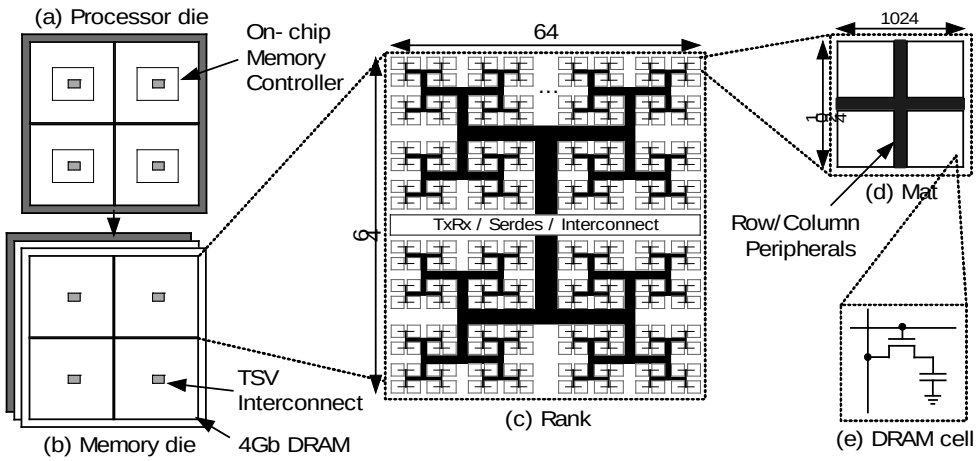


Figure 4.1: Overall layout of the exemplar system. (a) A processor die consists of 4 cores, each having a memory controller. Each memory controller is connected to 2 memory dies through TSVs. (b) Each memory die consists of 44Gb ranks. (c) A rank is partitioned to multiple mats connected through command and data networks having the H-tree topology. (d) A mat consists of 4 subarrays sharing a peripheral circuitry. (e) Each subarray has 1024 x 1024 DRAM cells.

Commodity DRAM chips have command/data networks with high fanins/fanouts that need sense amplifiers for fast and reliable data delivery. CACTI assumes H-tree networks for distributing commands and data to the mats. Since the H-tree is a binary tree, it needs more levels to connect the same number of mats but simple repeated wires can be used in the tree. We take the approach of CACTI for simplicity, but use energy-efficient wires that are the half the speed of the delay-optimal wires [73].

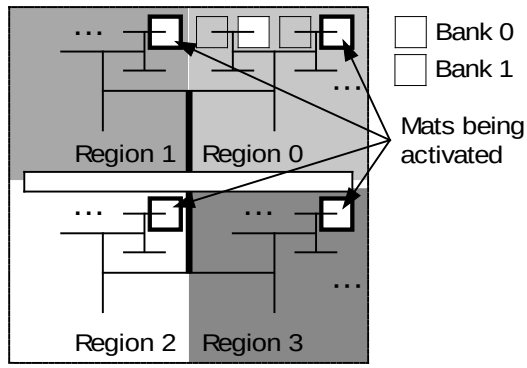


Figure 4.2: A distributed bank and page layout of a DRAM rank.

To improve the area efficiency of DRAM dies, minimal pitch wire style is assumed for the tree networks. Mat is a terminology used in CACTI, which is a memory structure with 4 subarrays, sense amplifiers and common decoders, where each subarray is composed of 2D array of DRAM cells.

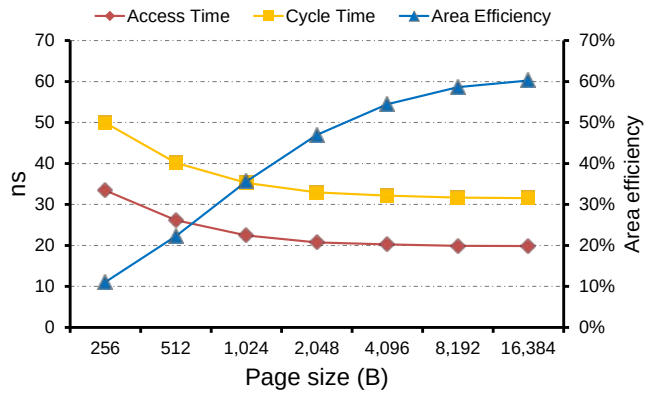
We modify the array organization of CACTI by distributing active mats and banks across an entire DRAM rank. 4 lines in a mat, one per subarray, are activated together, but usually their size is smaller than a DRAM page. CACTI introduces the concept of sub-bank, a block of consecutive mats, and assumes that all mats in a sub-bank are activated together. This organization helps limiting the power consumption of a command network that broadcasts to all the mats since all the active mats are located consecutively, but incurs area and latency overheads because the width of the data network outside of sub-banks must be the same as the number of bits transferred per column access nC.

Instead, we partition mats into R regions and make only one mat

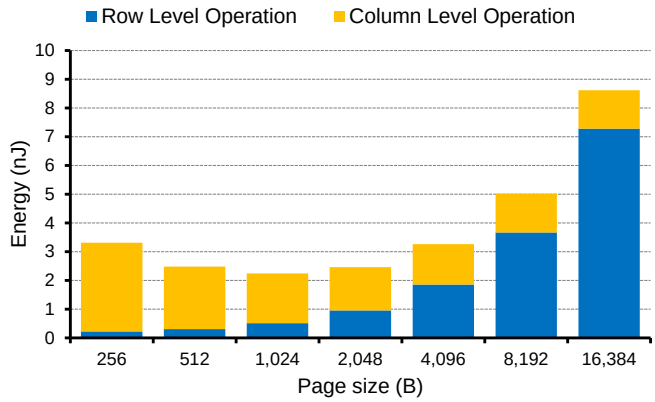
activated or precharged per row access, where R is the same as the number of mats per sub-bank per rank (Figure 4.2). In this scheme, data networks within a region broadcast to all the mats, and the width of the data network within a region is R times smaller than nC.

The width of the data networks doubles every time, two data networks get merged outside of the regions, so the area efficiency of the DRAM rank can be improved compared to the CACTI scheme, particularly when nC is large like on Wide I/O DRAMs. When a rank has multiple DRAM banks, they are also distributed across regions as shown in Figure 4.2, incurring negligible power, area, and timing overhead as far as the number of banks is smaller than the number of mats per region.

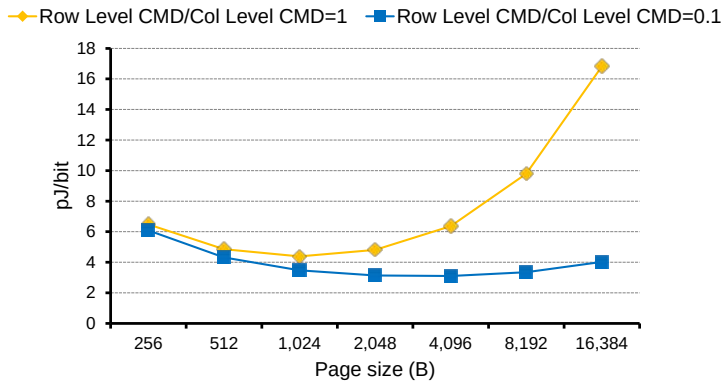
We run the modified CACTI to evaluate the impacts of varying the page size on the power, area, and timing of a DRAM rank. We configure CACTI to model a 32nm 4Gb rank, and fix the mat size to 1024 x 1024 for high area efficiency. The page size is varied from 256B to 32KB, which translates to 1 to 128 regions. The number of bits transferred per column access is 512, the size of a last-level cache line. Increasing the page size makes a page distributed across more mats with fewer bits per mat. As a result, the access latency and the cycle time get lowered leading to higher area efficiency as we increase the page size (Figure 4.3(a)). Figure 4.3(b) shows the energy consumed to activate and precharge a page and to read a cache line. When the page size increases, the energy to read a cache line is reduced, but the activate and precharge energy increases because more bitlines and sense amplifiers are used.



(a) Access time, cycle time, and area efficiency



(b) Activate/precharge energy and read energy



(c) Access energy efficiency

Figure 4.3: Power, area, and timing results of 4Gb main-memory DRAM bank modeled by the modified CACTI.

Since memory accesses with spatial locality can have several consecutive column accesses, we choose two distinguished row access to column access points to highlight the influence of varying the page size on different access patterns. Figure 4.3(c) shows energy to access a bit to DRAM. When each column access accompanies a row access, we achieve the highest energy efficiency when the page size is 1024 bits. If we increase the frequency of column accesses by 10 times, the sweet spot moves to 4096 bits.

4.2 Evaluation

To evaluate the impact of varying the page size and the number of banks per rank in the context of system-level performance and energy efficiency, we model a cache coherent chip-multithreaded processor system having a processor die and two DRAM dies.

4.2.1 Experimental Setup

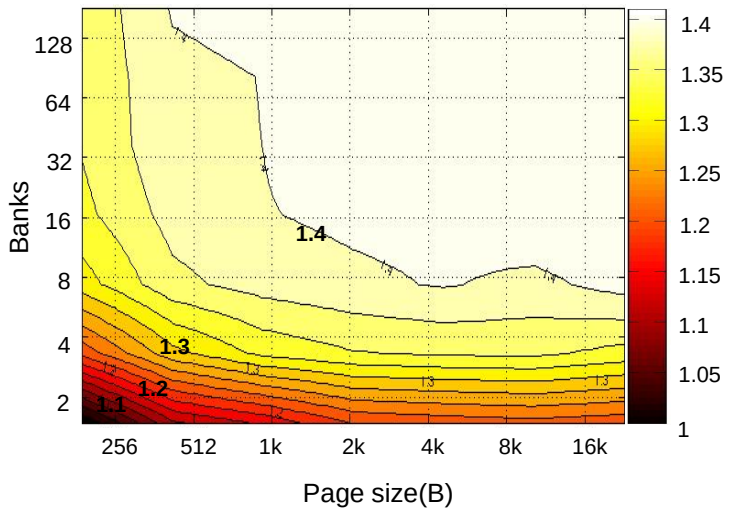
The processor die includes 16 in-order cores, and each core has 4 hardware threads. Each core has a separate 32KB L1 instruction and data caches. 4 cores share a 512KB L2 cache, a memory controller, and a directory constituting a cluster. The cache line size is 64B on all caches. Each memory controller is connected to two DRAM clusters, one per DRAM die, through TSVs. The scheduling

policy of the memory controllers is closed-page first-ready first-come-first-serve [11].

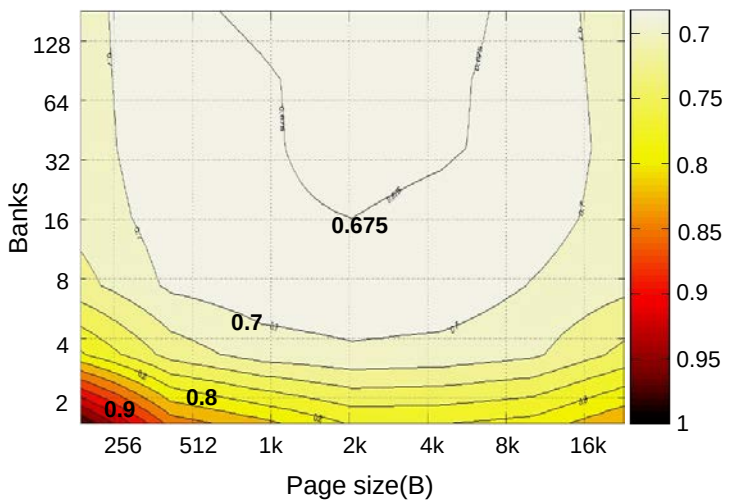
We use McSimA+, a manycore simulation infrastructure, for performance simulation, and McPAT [54] for power, area, and timing modeling of the processor die. We choose memory-intensive applications from the SPLASH2 [65] and SPEC CPU 2006 [74] benchmark suites. We run RADIX and FFT from SPLASH2 as multithreaded workloads, consolidate 433.milc, 450.soplex, 459.GemsFDTD, and 470.lbm into the multiprogrammed floating-point workload, and consolidate 429.mcf, 462.libquantum, 471.omentpp, and 473.astar into the multiprogrammed integer workload.

4.2.2 The System-level Impact of DRAM Array Organizations

Figure 4.4 shows the normalized instructions per cycle (IPC) and system energy-delay product (EDP) of the workloads. EDP is the product of the execution time of a benchmark and the energy consumed by the entire system including processors and main memory while the benchmark is being executed. As for the EDP, the lower is the better. We vary the page size from 256B to 16KB and the number of banks per rank from 2 to 128. Due to the limited space, we only show the average values of the 4 workloads described above. The IPC and EDP of each workload are normalized to the baseline configuration that has 2 banks and the page size of 256B.



(a) Normalized instructions per cycle (IPC)



(b) Normalized energy-delay product (EDP)

Figure 4.4: Average of the normalized IPC and EDP of the tested workloads on a chip-multiprocessing system.

When we fix the page size, both the IPC and the EPC are improved as the number of banks increases. This is because the number of banks is proportional to the number of active pages so

that the number of precharge and activate operations due to bank conflicts decreases incurring reduction in the execution time. If we vary the page size as well, we observe that the workload execution times are reduced as the page size increases but reach steady-states with about 40% increase in IPC on average against the baseline configuration when the number of banks is more than 8 and the page size is more than lKB.

Since the activate and precharge energy is mostly proportional to the page size and the execution times reach steady states, the EDP first decreases but later increases as we increase the page size. The EDP reaches the sweet spots when the page size is around 2KB and the number of banks is more than 8. Note that the Wide-I/O DRAM has 4 ranks per die, 4 banks per rank, and 2KB pages. Even though its capacity is different from our configurations, its design is close to the optimal configurations of our experiments in terms of the IPC and EDP.

Chapter 5

Silicon Interposer-Based Main Memory Systems

Modern microprocessors have adopted highly threaded multi-core designs to achieve energy-efficient performance with an increasing number of transistors on a die [75]. Applications have also evolved to execute multiple tasks to effectively utilize the increasing number of cores. However, the performance potential of a multi-core processor is realized only when the memory system can satisfy the increasing capacity and bandwidth requirements. While the increased transistor density can help increase memory capacity, it is much more challenging to scale memory bandwidth cost-effectively.

To address the increasing demands for processor-to-memory bandwidth, either the number of pins or the data rate of each pin

---

This chapter is based on [1]. - ©[2014] IEEE. Reprinted, with permission, from SC' 14.

must be increased. Modern multi-core processors have integrated an increasing number of memory controllers on a die [76], [77], [78] to increase the bandwidth by adding more pins, and leveraged DRAM or buffering devices running at a higher clock rate [79]. However, neither the pin transfer rates nor the number of pins can continue to scale easily. Boosting the pin transfer rates degrades the energy efficiency and signal integrity. Adding more pins increases the package area, which in turn increases the fabrication cost. Furthermore, assuming the energy per bit of inter-package data transfer is relatively constant, the power consumed in the memory system increases linearly to the number of pins, which makes memory channels consume a significant portion of the total package power.

To achieve high memory bandwidth without increasing the die area or off-package access energy, Through-Silicon Interposer (TSI)-based packaging provides an attractive alternative solution [9], [10] by combining two emerging technologies: 3D-stacked memory and interposer-based die integration. The interposer-based die integration connects a processor die and memory dies using in-package metal wires and Through-Silicon Vias (TSVs), while memory dies are stacked vertically using intra-die metal wires and inter-die TSVs. By using a low impedance and high bandwidth intra-package communication medium, TSI-based packaging is considered as a promising technology providing high memory bandwidth, high energy efficiency, and decent scalability in capacity.

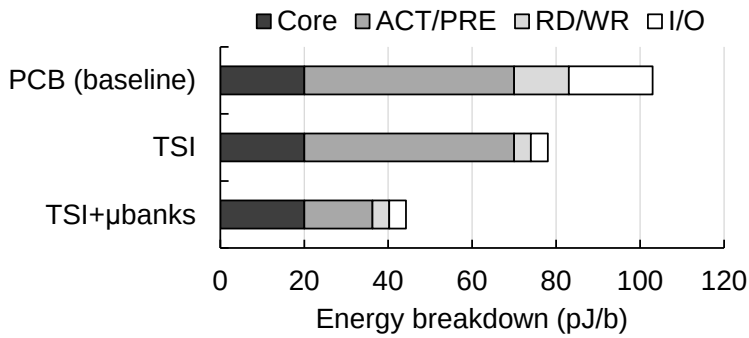


Figure 5.1: Energy breakdown of the conventional PCB-based, TSI-based, and proposed μbank-based memory system.

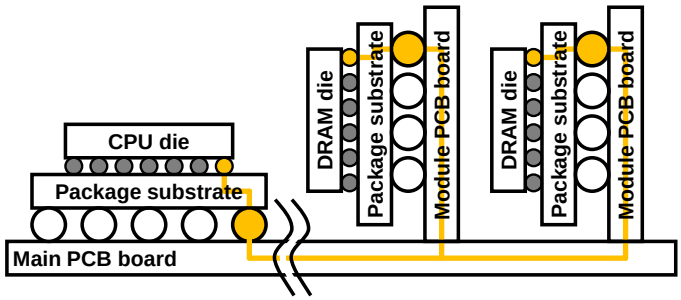
Processors with TSI-based integration (also known as 2.5D form factor) are expected to be introduced into the market before full 3D stacked processors without silicon interposers because of their advantages in cost and yield [80], [81]. Although there has been a significant amount of research done on TSV-based 3D integration, there have been limited architectural studies on TSI-based integration.

In this dissertation, we explore the impact of TSI technology on the design of the main memory system architecture. Compared with a baseline DDR3 main memory system where the components are connected through printed circuit boards (PCBs), the use of TSI in the memory system reduces the inter-package data transfer (I/O) power as shown in Figure 5.1. However, with a reduction in the I/O power dissipation, the energy consumed in the main memory system is "unbalanced" as the memory core energy consumption (e.g., activate/precharge energy) begins to dominate the overall energy consumption.

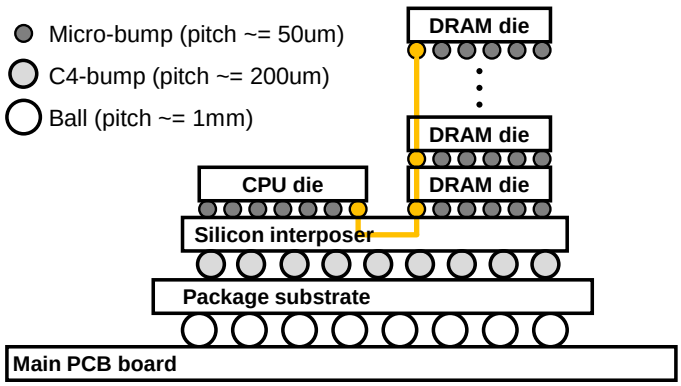
To address this, we propose *μbank*, a novel DRAM device architecture customized for TSI-based main memory systems. The μbank DRAM partitions each bank both horizontally and vertically into a large number of small banks (or μbanks) to activate fewer bits per activation, which corresponds to reducing the size of a DRAM page or row. The μbanks operate independently like conventional banks, and the partitioning granularity is chosen to minimize the area (cost) overhead while improving both the performance and energy efficiency.

While a larger number of μbanks significantly increases bank-level parallelism and improves energy efficiency, the effectiveness of prior approaches to memory system design needs to be re-examined. Because there are much fewer bank conflicts due to the larger number of open rows, a complex page-management policy is not necessary and hence μbank simplifies the DRAM controller design. Our evaluation shows that a sophisticated prediction-based page-management policy provides significant performance improvement on a conventional main memory system but a simple open-page policy [11] achieves comparable performance with μbanks. We also revisit the appropriate granularity of address interleaving and show that DRAM row interleaving outperforms cache-line interleaving because inter-thread interference mostly disappears with the massive number of banks.

We explain TSI technology in Section 5.1. Section 5.2 describes the motivation for μbank, and the detailed μbank architecture is shown in Section 5.3. Our evaluation appear in Section 5.4



(a) A conventional ball-grid-array-based system



(b) A through-silicon interposer-based system utilizing silicon interposer and through-silicon vias (TSVs)

Figure 5.2: Packaging technologies.

5.1 Through-Silicon Interposer (TSI)

In this section, we describe the Through-Silicon Interposer (TSI) technology. We then quantify the energy and performance benefits of the TSI technology when applied to the main memory system, compared with the conventional DIMM-based memory system.

5.1.1 TSI Technology Overview

The bandwidth density and energy consumption per data transaction

of inter-package communication through PCBs have improved very slowly compared to those of on-die computation and communication. An integrated circuit die, encapsulated in a package, is placed above and connected to a polymer-based substrate through a wire bonding or flip-chip process (Figure 5.2(a)). The substrate typically has an array of balls or pins whose pitch is around a millimeter and has not decreased substantially over time [12]. This is in contrast to the transistor pitch, which has improved much more rapidly following Moore's Law. Therefore, the number of pins per package has increased at a much slower rate compared with the computational capabilities, and system designers often rely on increasing the data transfer rate per I/O pin to alleviate the bandwidth pressure from on-die transistors.

An impedance mismatch caused by bulky substrate pads and balls as well as multiple wire stubs attached to the inter-die communication channel over PCBs results in the reflection of transmission waves, and hence poor signal integrity. Sophisticated impedance matching, such as on-die termination (ODT) and large drivers, is needed to deliver signals at a rate of multi-Gb/s per pin. If more than two pads are connected to an inter-die channel to increase the main memory capacity, the signal integrity of the channel gets degraded, leading to higher energy consumption per bit and even limiting the maximum data transfer rate.

For example, DDR3 interfaces [19] for servers and laptops support multiple ranks per memory channel but consume 20pJ/b. GDDR5 interfaces [82] for graphics support high bandwidth per pad

(up to 10 Gbps) but only allow point-to-point connections. LPDDR2 [83] interfaces for mobile systems have neither ODT nor delay-locked loops (DLLs), lowering the data transfer energy, but the data transfer rate per pin is heavily affected by the signal integrity and hardly surpasses one Gb/s. Therefore, bandwidth, energy efficiency, and capacity conflict with each other in the conventional inter-die processor-memory interfaces.

Through-Silicon Interposer (TSI) technology [9], [10] can address the bandwidth density (i.e., bandwidth per cross sectional distance, measured in Gbps/mm) and energy efficiency issues of the interconnects in processor-memory interfaces. A silicon interposer replaces a conventional polymer-based substrate and is located between a die and a substrate (Figure 5.2(b)). An interposer can be created with a method similar to fabricating a silicon die, but the process is simpler because it only has metal interconnect layers, vias, and pads; thus, it leads to lower costs. Multiple dies are attached to an interposer through *micro-bumps*, whose pitch is smaller than 50μm and an order of magnitude smaller than the ball (pin) pitch of the package. Even if micro-bumps were to be used between a die and a conventional substrate, the ball pitch of the package that contains the die and the substrate limits the benefits of the micro-bumps.

In contrast, the wire pitch of a silicon interposer can be as small as the pitch of the top-level metal layers of the silicon dies; therefore, it is possible to have thousands of inter-die communication channels using only one silicon interposer metal

layer. Escape routing [84] can be employed to resolve the pitch mismatch issue between the micro-bumps (few tens of microns) and the silicon-interposer wires (few microns [10]).

The channels over TSIs have a much better signal integrity than those over PCBs because the micro-bumps are smaller than the balls, and there are fewer wire stubs between the pads. With more channels between the dies, the data transfer rates per channel can be lowered while still providing high bandwidth, hence reducing the complexity of the transceivers and the I/O energy consumption per bit. Through Silicon Vias (TSVs) can be used to stack multiple dies, particularly low power DRAM dies, which effectively resolves the capacity problem in the main memory system.

5.1.2 The Energy Efficiency and Latency Impact of the TSI

To quantify the performance and energy efficiency impact of the TSI technology on processor-memory interfaces, we modeled inter-die I/Os by modifying CACTI-3DD [85]. To estimate the energy, area, and latency of a main memory DRAM device, we assumed a 28nm process, PTM low-power model [86] for the wires, and 3 metal layers. The minimal wire pitch of the global wordlines and datalines was conservatively assumed as 0.5μm [16] to reduce the inter-line interference; the pitch of the micro-bumps was 50μm, and that of the interposer wires was 5μm [87]. The capacity of a DRAM die was 8Gb, and the size of the baseline die was 80mm^2. The size of a DRAM page or row per rank was 8KB.

Energy Parameter		Value
I/O energy (DDR3-PCB)		20pJ/b
I/O energy (LPDDR-TSI)		4pJ/b
RD or WR energy without I/O (DDR3-PCB)		13pJ/b
RD or WR energy without I/O (LPDDR-TSI)		4pJ/b
ACT+PRE energy (8KB DRARM page)		30nJ
Timing Parameter	**Symbol**	**Value**
Activate to read energy	tRCD	14ns
Read to first data delay (DDR3)	tAA	14ns
Read to first data delay (TSI)	tAA	12ns
Activate to precharge delay	tRAS	35ns
Precharge command period	tRP	14ns

Table 5.1: DRAM energy and timing parameters.

Table 5.1 lists the modeled DRAM energy and timing values. In a DDR3 interface [19], which is the baseline that we assume in this work, the dominant portion of the read or write energy is the inter-die I/O energy, which is 20pJ/b [13], [88]. The energy to move data between bitline sense amplifiers and DRAM-side transceivers, which include local, global, and inter-bank datalines, is 13pJ/b.

A naive way to apply the TSI technology to the DDR3 interface is to vertically stack DRAM dies in a rank without modifying its physical layer. This can greatly improve the aggregate memory bandwidth of a processor because TSI eliminates the pin-count constraint. The energy efficiency, however, is only modestly improved because the DDR3 physical layer still has ODTs and DLLs

that draw considerable power.

A better way to exploit TSI is to replace the DDR3 interface with the LPDDR (low-power DDR) interface [83], [89]. The shorter physical distance in LPDDR obviates the need for ODTs and DLLs and significantly lowers the I/O and read/write energy. One issue with LPDDR is the lower per-pin transfer rate, but it can be overcome by increasing the number of pins exploiting TSI. Another issue with LPDDR is that the datapath is not delay-locked and jitter can become more problematic, especially across dies [87]. Therefore, we assume each die constitutes a rank, instead of using multiple dies. By applying the TSI technology and exploiting the LPDDR interface, the inter-die I/O energy efficiency improves substantially to be only 4pJ/b. We assume that a CPU-side pad and 8 DRAM-side pads constitute an inter-die channel, where high-speed serial links are not effective solutions.

However, the reduced I/O energy consumption leads to an "unbalanced" main memory design in terms of energy efficiency as the non-I/O portion (e.g., activate/precharge energy) begins to dominate the overall energy consumption. Modern performance-energy balanced cores need a few hundred pico joules (pJ) per operation [90], [91]. For example, a dual-issue out-of-order core, modeled by McPAT [54] (details in Section 5.4.1), consumes 200pJ/op in 22nm. Assuming 20 memory accesses per kilo-instructions (MAPKI) and a cache line size of 64B, each operation incurs $64 \times 8 \times 20/1000 = 10.24$ bits of data transfers from the main memory on average. Using the conventional interface, it translates

to 200pJ/op, which is on a par with the core energy consumption. By utilizing the TSI-based interface, only 40pJ is needed instead, which is much more energy efficient. Therefore, the improved energy efficiency of inter-die I/Os makes the activate and precharge energy more prominent, with their reduction becoming a key design challenge.

The impact of TSI on DRAM access latency is not as significant as the access energy. The internal structure of DRAM devices is mostly unaffected, and the latency of the inter-die channel is not high even in conventional interfaces (e.g., 170ps per inch [84]). However, a lower transfer rate per channel reduces the access latency because fewer serialization and deserialization steps are needed [92], [93]. The following section explores the customization of main memory system design to better exploit the opportunities opened by the TSI technology.

5.2 Microbank: A DRAM Device Organization for TSI-based Main Memory Systems

5.2.1 Motivation for Microbank

Energy to transfer a cache line through the processor-memory interface decreases substantially with the reduced I/O energy from the TSI technology. As a result, other components of DRAM power consumption, including static power (e.g. DLL and charge pumps),

refresh operations, and activate/precharge operations, represent a substantially higher fraction of total DRAM power. In particular, the energy to precharge the DRAM datapath and activate a row becomes 15× higher than the energy to read a cache line through interdie channels, as listed in Table 5.1. In this dissertation, we assume that a cache line is a unit of the main memory data transfers and is 64B.

One way to save the activate/precharge energy is to reduce the size of a DRAM row. Because the sub-wordline size of a mat is also 64 bytes (512 bits), the energy overhead from activate/precharge can be minimized by configuring a single mat to provide data for an entire cache line (referred to as a single subarray (SSA) configuration [71]). However, this results in significant DRAM die-area overhead because too many local datalines (i.e., 512) are needed per mat, which need to be routed in parallel with the wordlines. Because the number of metal layers is limited to a few due to extremely tight cost constraints in DRAMs (we assume 3 layers in this paper), the local datalines cannot be routed on top of the wordlines.

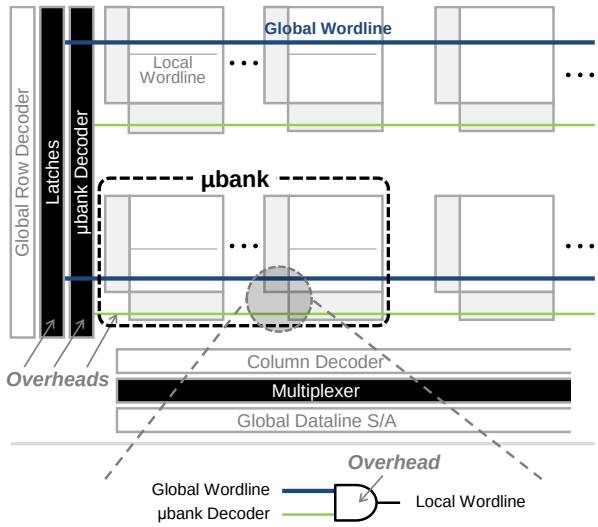
In addition, the pitch of these datalines is greater (0.5㎛ [71]) than the width or height of a DRAM cell in order to lower the resistance and capacitance. Therefore, the area of a DRAM die is increased by 3.8× with the SSA configuration compared to the reference DRAM die configuration in Section 5.1.2 and thus, makes this approach infeasible. To reduce the area overhead, we can activate multiple mats for a data transfer, which decreases the

number of bits transferred *per* mat, hence fewer local datalines are needed.

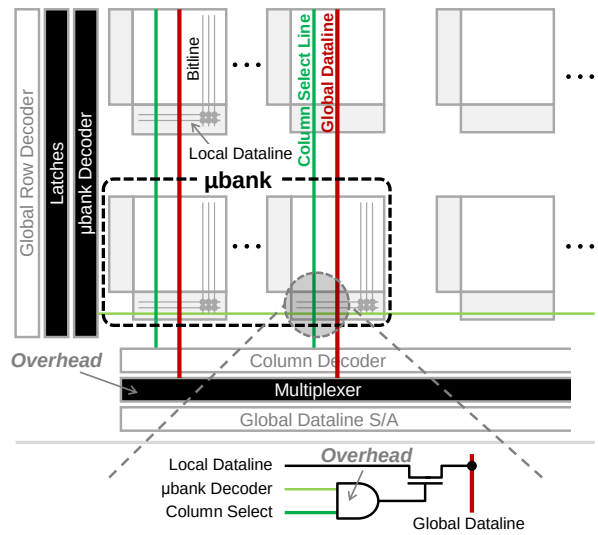
To increase the number of active rows without exploding the die area, we can exploit the bitline sense amplifiers within each mat to retain data. Note that, in conventional DRAM devices, the number of active rows is the same as the number of banks, and that increasing the number of banks incurs a high area overhead because global-dataline sense amplifiers and row/column decoders are bulky [94].

The overhead in exploiting the abundant bitline sense amplifiers is to add latches between row predecoders (global row decoders) and local row decoders to specify the row for the reads or writes [95] (Figure 5.3(a)). Further increasing the number of bitline sense amplifiers by decreasing the number of sub-wordlines per mat would incur a much higher area overhead because the size of a sense amplifier and related circuitry is an order of magnitude larger than a DRAM cell [88].

Thus, we group some number of physically adjacent mats and refer to them as a *μbank*, as shown in Figure 5.3. A μbank consists of a two-dimensional array of mats where only a row of mats can be activated. The number of active rows in a bank is equal to the number of μbanks. The additional μbank row and column decoders are used to identify the specific latches to be updated, and the signal from the μbank row decoder is again combined with the column select signals to specify the μbank used for the data transfers (Figure 5.3(b)).



(a) μbank row and column decoders select latches to update.



(b) μbank decoder and column select pick data to read and write.

Figure 5.3: μbank organization and operations. Compared to the baseline organization, μbank row/column decoders are added per bank. Latches are added to hold currently active wordline per μbank.

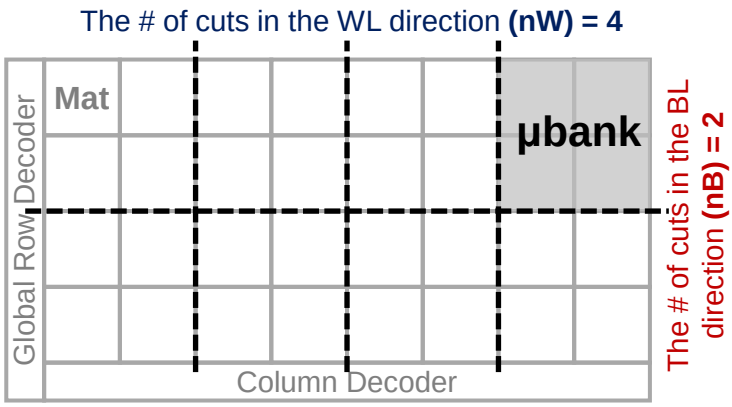
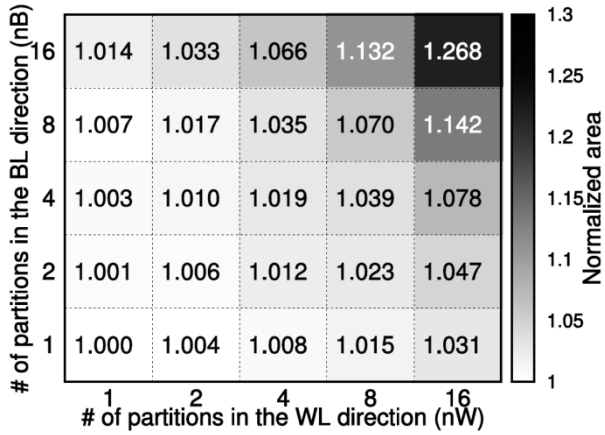
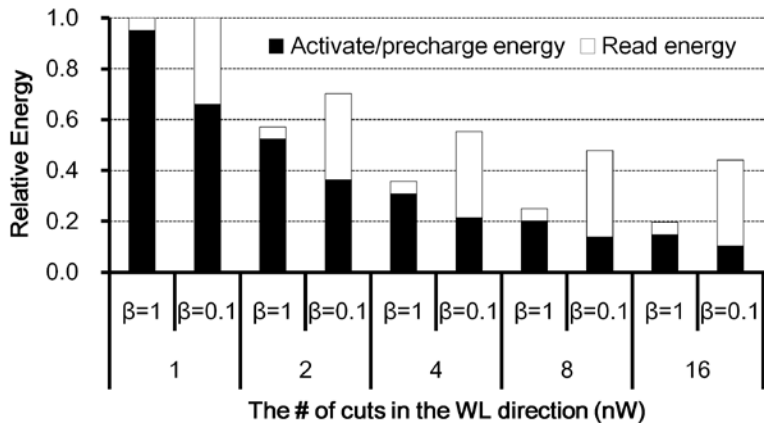


Figure 5.4: An example of a μbank design with (nW, nB) = (4,2). nW is the number of partitions in the wordline direction whereas nB is the number of partitions in the bitline direction.

The total number of μbanks within a bank is determined by the number of mats grouped together in both the wordline and bitline dimensions. We define nW as the number of bank partitions in the wordline direction, and nB as the number of bank partitions in the bitline or global dataline direction. Thus, the total number of μbanks per bank is $nW \times nB$. If $nW = nB = 1$, μbank is equivalent to a bank, which corresponds to the baseline in our evaluation. For example, for a bank consisting of 32 mats (Figure 5.4), it can be divided into four partitions in the wordline direction ($nW = 4$) and two partitions in the bitline direction ($nB = 2$) – thus, there are 8 μbanks per bank, each of which consists of 4 mats.



(a) Relative area



(b) Relative energy

Figure 5.5: The relative main memory DRAM area and energy while varying the number of partitions in the BL and WL directions. All the values are normalized to those of one μbank per bank values, respectively.

5.2.2 Microbank Overhead

The area and energy overhead of μbanks in a TSI-based main memory system are shown in Figure 5.5. We assume that an 8Gb DRAM die has 16 banks and 2 channels, where each channel serves 8 banks. The channel bandwidth is 16GB/s so that a 64B cache line

can be transferred every 4ns, which determines the internal clock frequency of the DRAM mats to be 250MHz. Each bank, whose size is 512Mb, consists of 2,048 512×512 mats, and is laid out as a 64×32 array. Figure 5.5(a) shows the area overhead of μbank. The x-axis is the number of partitions in the WL direction (nW), and the y-axis is the number of partitions in the BL direction (nB). The area overhead is normalized to $nW = nB = 1$, which has one μbank per bank.

Because of the added latches, the DRAM die area increases as the number of μbanks per bank increases. When $nW = 1$, 128 mats (2 rows of mats per bank to latch a 8KB row) are activated together, and each mat provides 4 bits of data out of the 512 activated cells within each mat per read or write to access 64B of data. For this configuration, routing 128 column select lines per mat incurs a noticeable area overhead because the pitch of a column select wire is greater than that of a DRAM cell [16].

Instead, we configure a column select line to choose 8 bits of bitlines and place a multiplexer between the 2 global datalines and the 1 global-dataline sense amplifier. As nW increases, while the number of global-dataline sense amplifiers stays unchanged, the total number of global datalines in a bank increases as the number of global datalines per μbank is fixed to the column width. Again, multiplexers are placed to select the right set of global datalines for the global-dataline sense amplifiers.

Meanwhile, the number of column select lines, which share the same metal layer with the global datalines, decreases for a reduced

number of columns per μbank. Therefore, compared to a baseline of $nW = 1$, the sum of the global datalines and the column select lines per bank does not increase as we increase the number of partitions in the wordline direction until 16. If we partition a bank into 16 pieces in both directions ($nW = nB = 16$), there is a 26.8% area overhead. However, for most of the other μbank configurations (when $nW \times nB < 64$), the area overhead is under 5%.

In addition to the area overhead, we quantify the energy overhead in Figure 5.5(b) and show the relative energy consumption of the DRAM configurations per read4 when the ratios of activate commands to read/write commands (β) are 1.0 and 0.1, respectively. If $\beta = 1$, it means that there is an activate command for every read/write command. As β decreases, the overhead of the activate/precharge operations is amortized over multiple read/write commands.

We normalize the energy consumption of each configuration to that of a single-μbank configuration for both values of β. As the number of μbanks per bank increases, more latches dissipate power, but their impact on the overall energy is negligible because the DRAM cells still account for a dominant portion of the bank area and power. The energy consumption per read is much more sensitive to the number of mats involved per activate command, where more μbanks in the wordline direction (nW) reduce the activate/precharge power, hence the energy per read. This is more prominent when β is high (i.e., low access locality).

5.3 Revisiting DRAM Page Management Policies

As the previous section focused on μbank-based memory systems tailored to the TSI technology, this section re-evaluates the effectiveness of conventional page-management schemes and then proposes a new scheme to exploit the larger number of banks available in the new memory systems.

Conventional memory controllers [11], [96] hold all pending memory requests in a queue and generate proper DRAM commands to service the requests, while obeying various timing constraints. The memory controllers can also schedule the requests and apply different page management policies to improve the per-bank row hit rates. For example, when the controller generates RD or WR commands to a specific bank, it can check the queue to find future memory requests that are also targeted to the bank. As long as the queue is not empty, the controller can make an effective decision of closing the row or keeping it open.

The two most basic page-management policies either keep the page open (activated) expecting for the next access' s row hit (i.e., open-page policy) or close it immediately expecting for a row miss (i.e., close-page policy) [11]. The more sophisticated, adaptive policies include minimalist-open policy [23] to close a row after observing a small number of row hits, and reinforcement learning (RL) approaches [21], [22] to adapt the scheduling policy based on the access history in the past. However, if the queue is empty, the controller must manage the page speculatively.

In fact, there exist two factors that make it difficult to employ these conventional memory scheduling policies to μbank-based memory systems, which require future memory requests available in the request queue (i.e., pending requests.) First, a memory request stream is now distributed over a larger number of banks in μbank-based memory systems compared with conventional memory systems, hence decreasing the average number of pending requests per bank. Second, the higher channel bandwidth provided by μbank-based memory systems further reduces the average queue occupancy. As a result, the request queues in μbank-based memory systems are very likely to fail in providing the information of future memory requests to the memory controller so that it cannot make an effective decision to manage the pages.

Therefore, for μbank-based memory systems, we devise a *prediction-based* page management scheme to adapt between close-page and open-page policies, based on the history of past memory requests. In this way, the new page management scheme can make effective page-management decisions without examining future memory requests in the queue. Our example design is based on a standard 2-bit bimodal branch predictor, which tracks the prediction results with either open or close (instead of taken or not taken) for each bank. The 2-bit predictor utilizes four states (00: strongly open, 01: open, 10: close, 11: strongly close), in which the two open states result in "predict open" page policies and the two close states result in "predict close" page policies. Depending on the accuracy of the previous prediction, the state is changed

accordingly, similar to conventional branch predictors. However, there exist several differences between conventional branch predictors and the page-management policy predictor, which can affect the effectiveness of the prediction differently. For example, even though conventional branch predictors resolve branches in several clock cycles, the page-management predictor may take much longer (e.g., several milliseconds.) In addition, address aliasing is much less of an issue in the page-management predictor because the number of DRAM pages is much smaller than the program's address space.

5.4 Evaluation

We simulated a chip-multiprocessor system with multiple memory channels to evaluate the system-level impact of the μbank-based main memory systems exploiting the TSI technology (Figure 5.6).

5.4.1 Experimental Setup

We assumed a system with 64 out-of-order cores, each running at 2GHz. Each core issues and commits up to two instructions per cycle, has a 32-entry reorder buffer, and has separate L1 I/D caches. Four cores share an L2 cache. We set the size and associativity of each L1 cache and L2 cache to 16KB and four, and 2MB and 16, respectively.

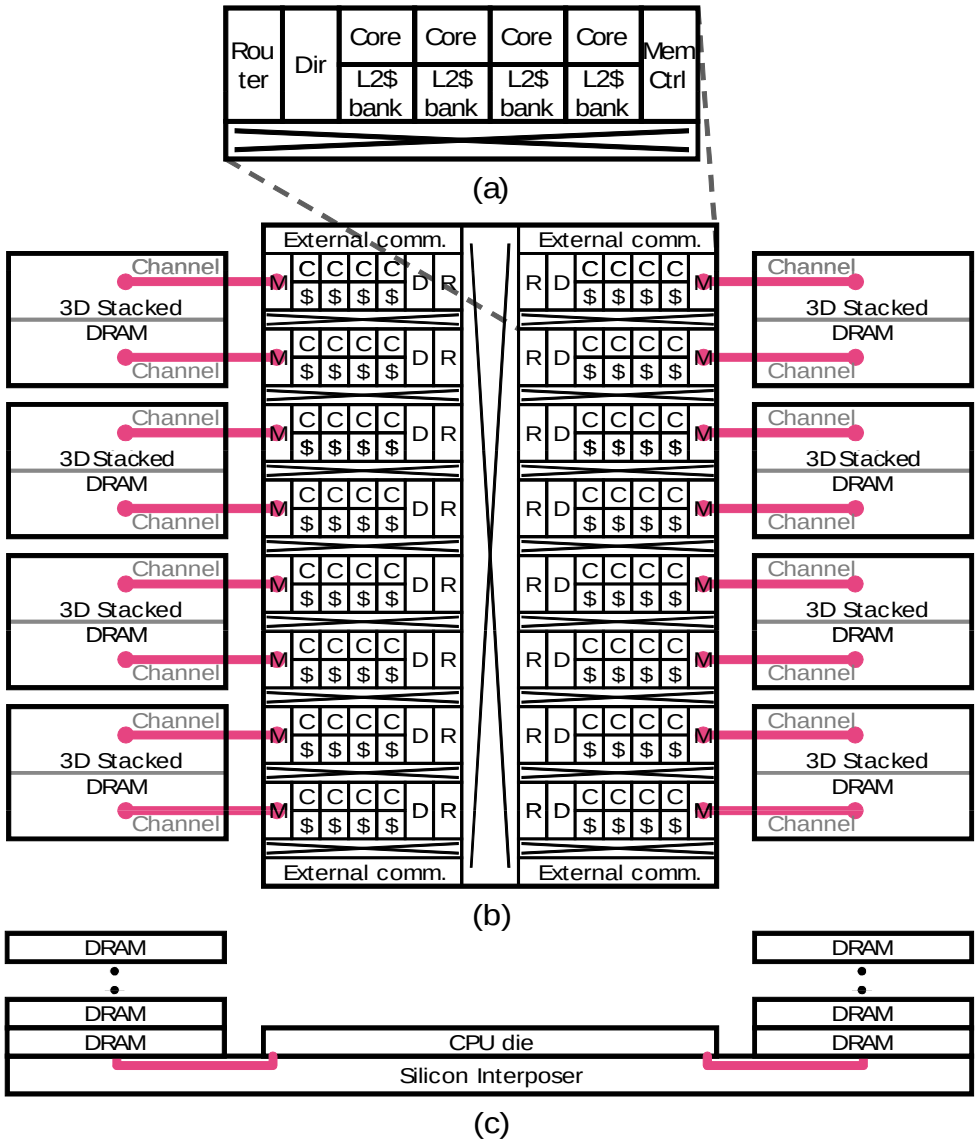


Figure 5.6: A 64-core chip-multiprocessor system with 16 clusters. (a) Each cluster has four cores (C), an L2 cache ($), a directory unit (D), a memory controller (M), and a router (R). A floorplan is shown in (b) and a cross-section view of the system is shown in (c).

Each cache is partitioned into four banks, and the line size of all the caches is 64B. The system uses a MESI cache-coherency protocol, and a reverse directory is associated with each memory controller. Figure 5.6(a) shows that each cluster consists of four cores, an L2 cache, a directory unit, a memory controller, and a router. The system has 16 memory controllers, and each controller has one memory channel, whose bandwidth is 16GB/s excluding the ECC bandwidth.

To evaluate single-threaded programs, we populated only one memory controller for the simulated system to stress the main memory bandwidth. Each memory controller has a 32-entry request queue by default, and applies PAR-BS [20] and the open-page [11] policy for memory access scheduling. The main memory capacity is 64GB. We modified McSimA+ to model the μbank-based main memory systems. We used McPAT [54] to model the core/cache power, area, and timing. The power, area, and timing values of the processor-memory interfaces and the main memory DRAM devices were modeled as explained earlier in Section 5.1.2 and summarized in Table 5.1.

5.4.2 Test Workloads

For the evaluation, we used the SPLASH-2 [65], SPEC CPU2006 [74], PARSEC [97], and TPC-C/H [98] benchmark suites. We used Simpoint [99] to identify the representative phases of the SPEC CPU2006 applications.

Group	SPEC CPU2006 applications
spec-high	429.mcf, 433.milc, 437.leslie3d, 450.soplex, 459.GemsFDTD, 462.libquantum, 470.lbm, 471.omnetpp, 482.sphinx3
spec-med	403.gcc, 410.bwaves, 434.zeusmp, 436.cactusADM, 458.sjeng, 464.h264ref, 465.tonto, 473.astar, 481.wrf, 483.xalancbmk
spec-low	400.perlbench, 401.bzip2, 416.gamess, 435.gromacs, 444.namd, 445.gobmk, 447.dealII, 453.povray, 454.calculix, 456.hmmer

Table 5.2: We categorized the SPEC CPU2006 applications into 3 groups depending on the number of main memory accesses per kilo instructions (MAPKIs).

Per SPEC application, we chose the top-4 slices in weight, each having 100 million instructions. As for SPLASH-2 and PARSEC, we simulated regions of interest and used the datasets listed in [54]. As for server workloads, we used two database workloads (TPC-C/H) from TPC benchmark. We carefully tuned PostgresSQL DB [100] to populate the target system with the database workloads.

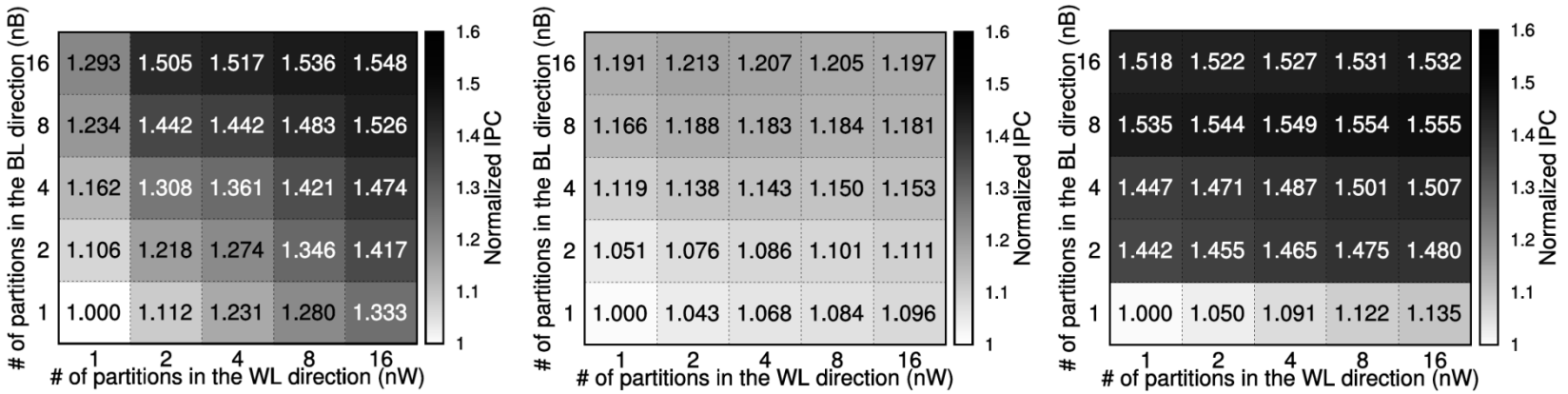
We classified the SPEC CPU2006 applications into three groups based on the main memory accesses per kilo-instructions (MAPKI) [69] (Table 5.2). We created two mixtures of multiprogrammed workloads: mix-high from spec-high applications and mix-blend

from all three groups. Per mixture, a simulation point is assigned to each core, and the number of populated points is proportional to their weights.

5.4.3 The System-level Impact of the μbanks

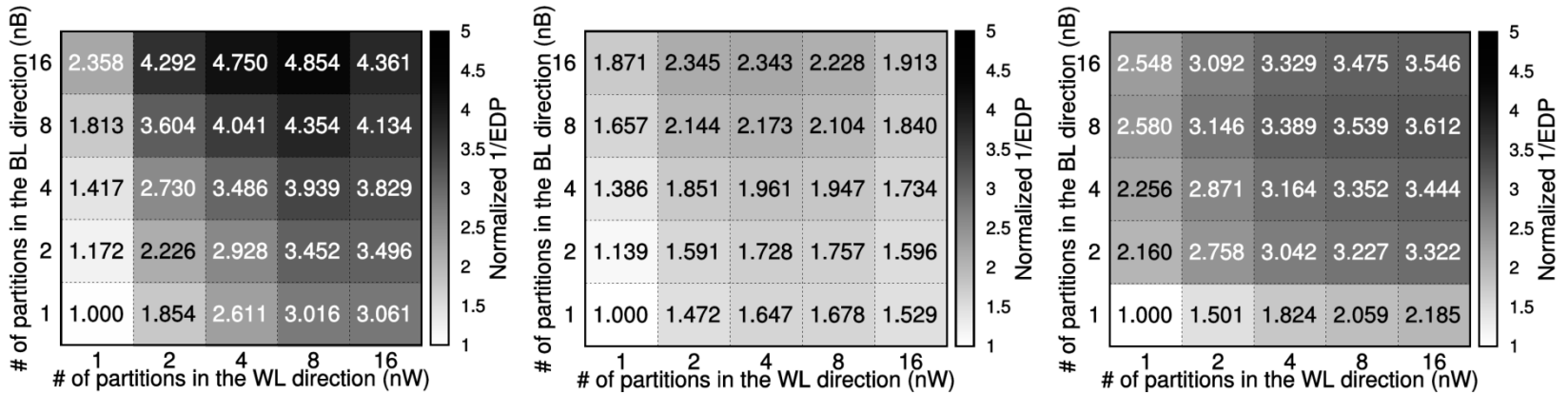
We first show that dividing a bank in either bitline or wordlines directions improves the performance with diminishing returns as the number of μbanks increases. In general, the bitline-direction partitioning (changing *nB*) yields higher returns on investment. For these experiments, we ran bandwidth-intensive SPEC CPU2006 applications and database workloads to estimate the performance and energy-efficiency gains on the die area increase with adopting μbanks.

Figure 5.7 shows the relative IPC values of 429.mcf, the average for the spec-high applications, and TPC-H. The unpartitioned configuration ((*nW*, *nB*) = (1, 1)) is the baseline. When a global dataline traverses more μbanks (higher *nB*), the IPC increases because there are more active rows while the row size is unchanged. As a global wordline traverses more μbanks (higher *nW*, the row size becoming 8KB/*nW*), IPC steadily increases on low *nB* values. However, creating many wordline partitions on high *nB* values can reduce IPC improvements because more active rows become available for higher *nW*, but each row gets smaller, underutilizing spatial locality in a request stream.



(a) 429.mcf (b) the average of spec-high (c) TPC-H

Figure 5.7: The relative IPC of (a) 429.mcf, (b) spec-high, and (c) TPC-H. Baseline is (nW, nB) = (1,1) for each.



(a) 429.mcf (b) the average of spec-high (c) TPC-H

Figure 5.8: The relative 1/EDP of (a) 429.mcf, (b) spec-high, and (c) TPC-H. Baseline is (nW, nB) = (1,1) for each.

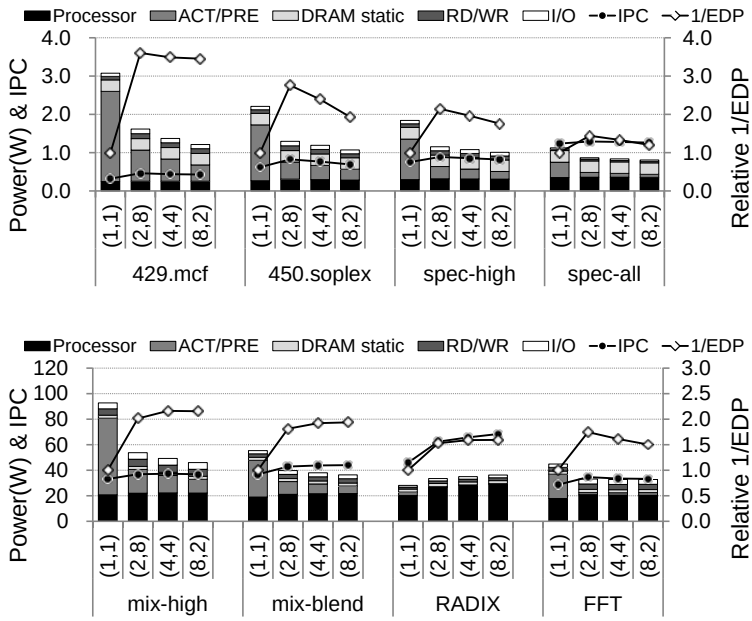


Figure 5.9: The relative IPC, 1/EDP, and power breakdown of applications on representative μbank configurations. spec-all stands for the average of all single-threaded SPEC CPU2006 applications. (nW, nB) = (1,1) is the baseline.

On 429.mcf, *nB* and *nW* had a similar impact on performance, and (*nW*, *nB*) = (16, 16) performs the best, for which the IPC is 54.8% higher than the baseline. TPC-H is more sensitive to *nB* than *nW*, and the configuration that gives the highest performance is (*nW*, *nB*) = (16, 8). We observed similar trends in the energy-delay product (EDP) metric.

Figure 5.8 shows the relative 1/EDP of 429.mcf, spec-high, and TPC-H. As we present the reciprocal of the relative EDP, a higher value indicates better energy efficiency. First, 429.mcf achieves the

highest IPC and 1/EDP values in (nW, nB) = (16, 16) and (8, 16) configurations, respectively. However, TPC-H and spec-high achieve the highest IPC and 1/EDP values in (16, 8) and (2, 16) configurations, respectively.

These results indicate that the merit of a lower activate/precharge energy for a smaller row size can outweigh the overhead of more ACT/PRE commands. Therefore, a balanced approach is necessary in increasing the number of active rows and reducing the size of the rows under a given area constraint. The configurations with more μbanks is also very effective when we consider the area overhead. Because the DRAM industry is highly sensitive to die area (hence cost), we chose the configurations with an area overhead less than 3% in this experiment, but achieving the most of IPC and EDP benefits.

Figure 5.9 shows the relative IPC and 1/EDP, and power breakdown of single-threaded, multiprogrammed, and multithreaded applications on the representative μbank configurations. The ones with more partitions in the wordline direction dissipate less activate/precharge power. The IPC and EDP values are improved more in memory-bandwidth intensive applications, which have high MAPKI values. In particular, RADIX, a SPLASH-2 multithreaded application, has high MAPKI values and row-hit rates for μbank-based systems, whose IPC value improves by 48.9% on (nW, nB) = (8, 2) configuration.

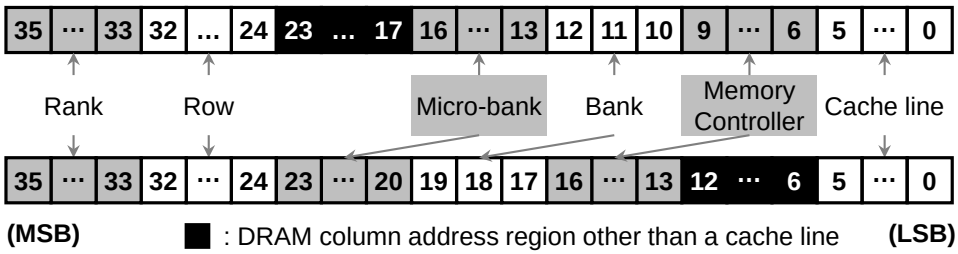


Figure 5.10: Two of the possible address interleaving schemes for (nW, nB) = (2, 8).

5.4.4 The Impact of Address Interleaving and Prediction Based Page-Management Schemes on μbanks

More active rows offered by the μbank microarchitecture can alter the optimal address mapping for DRAM accesses, such as the location of the address interleaving base bit (iB). The page management schemes also need to be reevaluated as discussed in Section 5.3. An example of alternative address interleaving is shown in Figure 5.10.

Existing memory controllers [23], [95] often choose the (micro-)bank number from the low significant bits of memory addresses such that the address interleaving is done at the granularity of one or few cache lines instead of a DRAM row granularity. However, the average row-buffer hit rate for the address interleaving at a DRAM row granularity can increase substantially as the number of active rows increases. Therefore, combined with the open-page policy, a page-granularity

interleaving could be more beneficial to μbank than cache line-interleaving.

On the other hand, with fewer active rows, the close-page policy with cache-line interleaving (iB = 6) can perform better than the open-page policy with page-interleaving. To evaluate the impact, we varied iB from 6 to 13 and chose (nW, nB) = (1, 1), (2, 8), (4, 4), and (8, 2) configurations (Figure 5.11).

The baseline configuration is (nW, nB) = (1, 1), open-page policy, and page interleaving (iB = 13). First, with fewer active rows, the difference between the two is not much for both IPC and 1/EDP. This is because the memory access scheduler (we used PAR-BS [20]) detects and restores spatial locality that can be extracted from the request queue of the memory controller. Second, the open-page policy with page-interleaving has greater advantage with the increased number of active rows.

For example, with 16 times more available active rows, the open-page policy with page-interleaving clearly outperforms the close-page policy, as high as 17.2% on spec-high for (nW, nB) = (2, 8). This is due to the spatial locality in main memory accesses that was hidden by the intra-thread and inter-thread interference, but is now successfully restored by more active rows.

The prediction-based page-management schemes consistently provide performance gains over fixed management schemes for applications, but only modestly on average. In this experiment, we evaluated three prediction-based page management schemes - local (per bank history bimodal) prediction, global (per thread

history bimodal) prediction, and tournament-based prediction schemes. As for the tournament scheme, we applied a bimodal scheme to pick one out of the open, close, local, and global predictors. We treated the open and close-page management polices as static predictors. We implemented them on top of the default out-of-order scheduler (PAR-BS [20]) to recover access locality to each bank across multiple interleaved request streams.

Our key finding is that the simple, static open-row policy achieves comparable performance with the prediction-based policies, which obviates the needs for complex page-management policies in μbank-based memory systems. Figure 5.12 shows the relative IPC and 1/EDP values and the predictor hit rates of the prediction schemes for (nW, nB) = (1, 1), (2, 8), and (4, 4). 429.mcf has a lower spatial locality in main memory accesses while canneal has a higher spatial locality than the average of the spec-high applications.

Therefore, the close-page policy has a higher prediction hit rate and better performance than the open-page policy on 429.mcf, and vice versa on canneal. Note that the local prediction scheme has a higher hit rate than both static policies, open and close, contributing to the highest hit rate and IPC of the tournament predictor. In the experiments, the global predictor never performs the best, so it is neither presented in Figure 5.12 nor considered as a candidate of the tournament predictor.

In many applications, the open-page policy outperforms the close-page policy and performs on a par with the tournament

predictor scheme because μbanks sufficiently provide many activated rows such that the prediction hit rate of the open-page policy is as high as that of the tournament predictor. The tournament predictor performs better than the open-page policy by 3.9% on average, and up to 11.2% on 429.mcf for (nW, nB) = (2, 8). In the future, for workloads with more complex access patterns, the cost of the tournament predictor may be justified.

5.4.5 The Impact of OS Page Migration

Most operating systems (OS) allocate and deallocate the main memory at the page granularity. A page is – typically 4KB [120] in size and is a chunk of main memory managed by the OS. Most modern OSes also support 2MB or 1GB pages in order to reduce TLB misses [120]. Pages are initialized or copied by the OS. The OS zeros deallocated pages for security reasons, because contents like personal IDs or passwords, can be exposed to malicious users [121][122]. Large-scale systems periodically checkpoint the memory footprints of processes to recover them in case of failure.

Figure 5.11 shows the OS page copy overhead of the representative μbank configurations. We use a modified STREAM benchmark [125] to model the OS page copy, which sequentially copies from the contents of a row in a bank to a row in another bank. We observed that partitioning a bank in the nW direction rarely incurs performance and energy overhead since the OS pages are distributed to multiple μbanks and accessed concurrently.

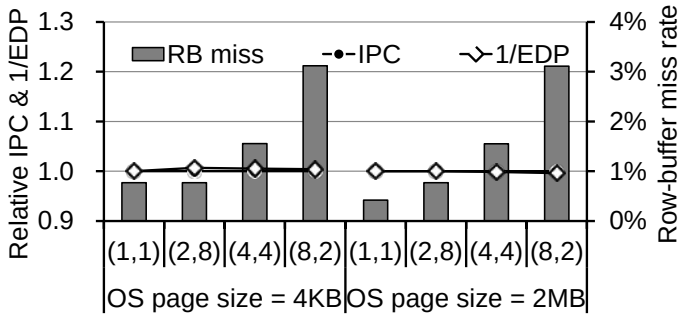


Figure 5.11: The IPC, relative EDP, and row-buffer miss rate of the representative μbank configurations.

When the OS page size is 4KB, (*nW, nB*) = (1, 1) and (2, 8) shows the same row buffer miss rate because the page size of (2, 8) is also 4KB. However, the row buffer miss rate increases when *nW* is larger than 2. The read and write energy dominates the overall energy consumption of memory systems, because the row buffer miss rate is below 4% regardless of μbank configurations and OS page size. The difference of performance and energy efficiency between *(nW, nB)* = (1, 1) and other μbank configurations is below 1%.

5.4.6 The Impact of DRAM Refresh

Every DRAM cell must be periodically rewritten to prevent data corruption due to charge leakage through a process called *refresh* [93]. Modern memory systems issue DRAM refresh commands at rank granularity [19] [84]. DDR3 DRAM devices refresh all banks with a single refresh command [19]. LPDDR2 also supports per-bank refresh where the command refreshes a single bank at a time[126].

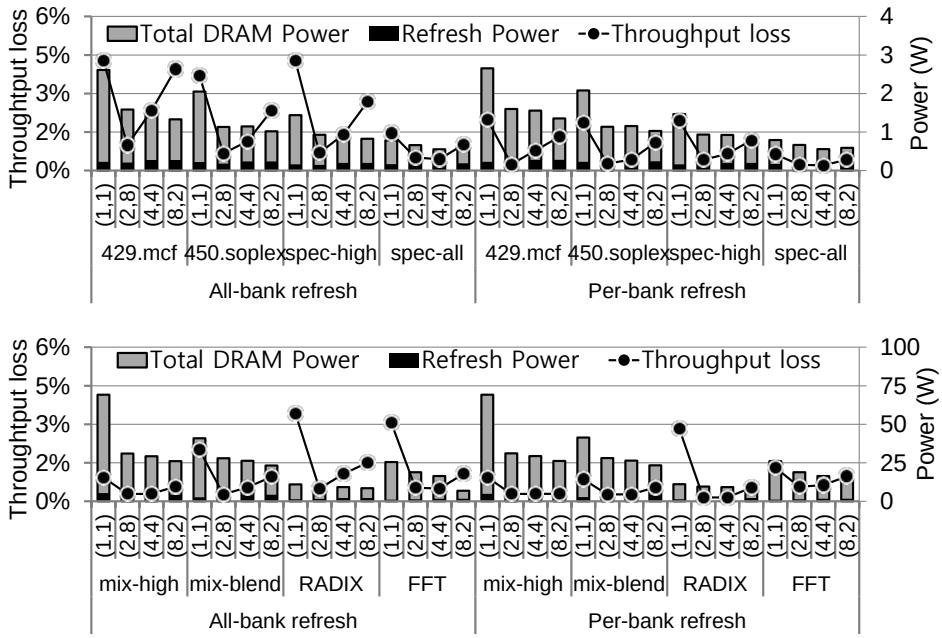


Figure 5.12: The performance overhead and DRAM refresh power of the representative μbank configurations.

The refresh interval (tRFI) of 4Gb LPDDR2 is 3.8μs in the all-bank refresh mode but tRFI becomes 1/8 in the per-bank refresh mode [126]. The refresh overhead of the per-bank refresh is smaller than the all-bank refresh because the refresh cycle time (tRFC) of the per-bank refresh is 60ns while that of the all-bank refresh is 130ns [126]. To minimize the refresh overhead of μbank-based memory systems, a group of μbanks in the nW direction is refreshed together with a single refresh commands. Consequently, all μbank configurations have the same refresh overhead. Because the single rank capacity of every μbank configuration is 4Gb, a single row refresh is sufficent for each bank.

Figure 5.12 shows the performance and energy efficiency loss

due to DRAM refresh for each representative μbank configuration. The per-bank refresh provides a better performance than all-bank refresh due to shorer tRFC. In RADIX, the performance loss of (*nW, nB*) = (1,1) becomes 3.42% in the all-bank refresh mode while that of *(nW, nB) = (2, 8)* shows 0.49%. The DRAM refresh has higher performance penalty when the number of banks is small. Regardless of DRAM refresh modes, dividing a bank in the *nW* direction increases the performance loss since the μbanks partitioned in the *nW* direction are refreshed in parallel. Additionaly, the energy overhead due to DRAM refresh also increases, because the partitoned DRAM row decreases activation and precharge energy while the energy consumption of DRAM refresh does not change regardless of μbank configurations. In single-high, the enrgy overheads of per-bank refresh in (*nW, nB*) = (1, 1), (2, 8), (4, 4), and (8, 2) are 11.09%, 16.02%, 21.53%, and 23.90% respectively.

5.4.7 The Impact of TSI on Processor-Memory Interfaces

To quantify the performance and energy benefits of the TSI-based processor-memory interfaces *without* μbanks, we compared three interfaces: module-based DDR3 connected through PCBs (DDR3-PCB), TSV-based stacked DDR3-type dies connected through a silicon interposer (DDR3-TSI), and TSV-based stacked LPDDR-type dies connected through a silicon interposer (LPDDR-TSI).

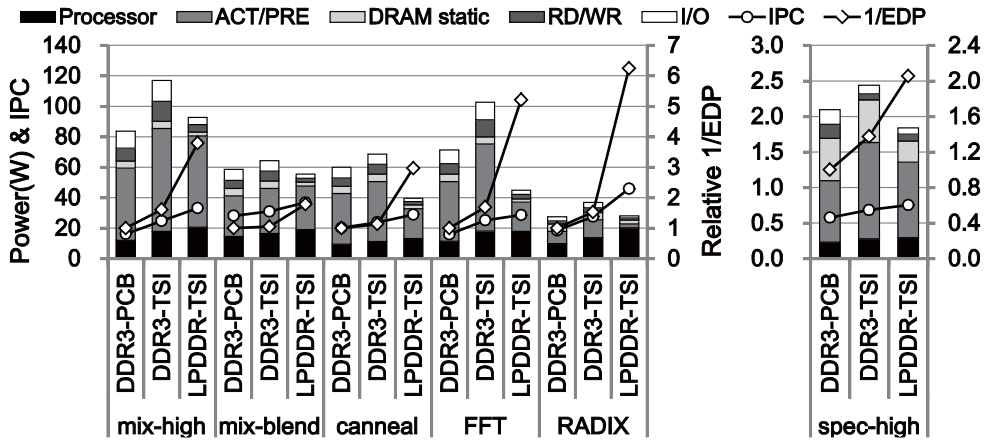


Figure 5.13: The IPC, power, and relative EDP of 3 processor-memory interfaces (DDR3-PCB, DDR3-TSI, and LPDDR-TSI) on multiprogrammed, multithreaded, and the average of spec-high workloads.

Figure 5.13 shows the IPC values, power breakdowns, and relative EDP values of the interfaces on multiprogrammed (mix-high and mix-blend) and multithreaded (FFT, RADIX, and canneal) workloads. For DDR3-PCB, we used eight memory controllers to keep their I/O pin count realistic (around 1,600 pins). For DDR3-TSI, a rank consists of eight DRAM dies. All configurations have the DRAM row size of 8KB. Exploiting TSI improves both the performance and energy efficiency even on conventional DDR3 interfaces, while adopting low-power processor-memory interfaces further saves main memory access energy.

For mix-high, DDR3-TSI and LPDDR-TSI achieve 52.5% and 104.3% higher IPC, and 37.8% and 73.7% lower EDP than that of DDR3-PCB, respectively. For LPDDR-TSI, the relative portion of

the activate and precharge (ACT/PRE) power out of the total memory power increases to 76.2% for mix-high, and thus, reducing the ACT/PRE power becomes the primary goal of μbank.

5.5 Related Work

Die Stacking Technologies

Die stacking technologies have recently received much attention to improve the throughput, latency, and energy efficiency of main memory systems. Virtex-7 FPGAs from Xilinx use stacked silicon interconnect (SSI) technology, which combines multiple FPGA dies using TSIs within a single package [81]. Nvidia has announced that its future Volta GPU will integrate stacked DRAM and a GPU into a single package in a similar way [101].

IBM and Sony have also announced their plans to adopt the TSI technology to scale Moore's Law with Module on Interposer [102] and High-Bandwidth Memory (HBM) [56]. While TSI-based integration technology is a promising option to provide energy-efficient high-performance main memory, the architectural implications of this technology are much less studied than full 3D stacking technologies [103], [104], [105].

Hybrid Memory Cube (HMC) [56] stacks multiple DRAM dies on top of a logic die and the CPU communicates with the HMCs through high-speed serial links. TSI replaces these links with lower-speed parallel interposer wires. As a result, the HMC has a

higher latency and static power and is not necessarily more energy-efficient for the system size being considered (e.g., single-socket system). An HMC-TSI hybrid approach would be an interesting approach because the system size is scaled up, but we leave it as part of future work.

DRAM Microarchitectures and Systems

Researchers have proposed various modifications to the conventional DRAM organization to improve performance and energy efficiency. Kim et al. [95] propose the subarray-level parallelism system that hides the access latency by overlapping multiple requests to the same bank. Gulur et al. [106] replace the existing single large row buffer with multiple sub-row buffers to improve row buffer utilization. μbank introduced in this paper subsumes both designs in that it partitions each bank along both bitlines and wordlines. This work was done in parallel with Half-DRAM [107], which also exploits vertical and horizontal partitioning of the conventional bank structure.

However, Half-DRAM applies partitioning in a conventional processor-memory interface and they do not discuss how to exploit the massive number of row buffers. Tiered-Latency DRAM [94], CHARM DRAM [88], and row-buffer decoupling [17] reorganize a DRAM mat to lower the access time for an entire mat or its portion. Although they are introduced in the context of the conventional non-stacking DRAM system, they are complementary and applicable to the μbank devices as well. Alternatively, there

are DRAM system-level solutions that need not modify the DRAM device organization.

Sudan et al. [108] propose micro-pages which allow chunks from different pages to be co-located in a row buffer to improve both the access time and energy efficiency by reducing the waste of overfetching. Rank subsetting [6], [71], [89], [7] is a technique that utilizes a subset of DRAM chips in a rank to activate fewer DRAM cells per access and improves energy efficiency or reliability. Unlike these system-level techniques, μbank is a device-level solution that either obviates the need for these techniques or complements them.

DRAM Access Scheduling

As discussed in Section V, most of the DRAM controllers that have been proposed [96], [109] make a decision of whether to leave a row buffer open or not after a column access (open-page vs. close-page) by inspecting future requests waiting in the request queue. Kaseridis et al. [23] propose the Minimalist Open scheme, which keeps the row buffer open only for a predetermined time interval (tRC). Piranha [110] takes a similar approach with a different interval (1μs). To improve long-term adaptivity, both Ipek et al. [21] and Mukundan et al [22] take a reinforcement learning (RL) approach to optimize the scheduling policy by considering the access history in the past.

Although the implementation details of the memory controllers are not available, Intel implements an Adaptive Page Management

(APM) Technology [24] while AMD has a prediction mechanism that determines when to deactivate open pages based on the history of the particular page [25]. Our results show that with the large number of banks, a simple open-page policy can be sufficient to simplify memory controller design.

Chapter 6

Conclusion

In this dissertation, we explored the main memory system architecture for modern manycore systems. To this end, we presented McSimA+, a cycle-level simulator to satisfy the requirements of manycore microarchitecture research. McSimA+ supports emerging asymmetric manycore systems, provides the detailed microarchitecture models for core and uncore subsystems, and supports more than 1,000 cores. As a case study, we proposed a novel asymmetric manycore design, Asymmetry Within a cluster Symmetry Between clusters (AWSB), to mitigate thread migration overhead by forming a cluster with different cores. Our AWSB manycore system shows noticeable improvement on performance compared to a state-of-art SWAB-style manycore system.

Then, we explored modern DRAM array organizations by varying the number of banks and DRAM row size of each bank. The modified CACTI results show that larger DRAM rows improve area

efficiency and access time, but degrade energy efficiency because of more activate/precharge energy consumption. We evaluated the system-level impacts of DRAM array organizations by simulating a manycore system with 3D stacked memory. The performance and energy efficiency of the tested manycore system were improved as we increased the number of banks, and 2KB DRAM row showed the best system-level energy-delay product. With the understanding of modern DRAM array organizations and manycore system architecture, we proposed a new TSI-based main memory system for manycore systems, which utilized a novel DRAM device microarchitecture, called *μbank*. We show that conventional DRAM cores dominated the overall energy consumption in TSI-based memory systems. To address this problem, we introduced μbank DRAM which partitions each conventional bank into a large number of smaller banks (or μbank) with small area overhead, under 5% of a DRAM chip area.

As a massive number of μbanks provide less activate-precharge energy whereas ample bank-level parallelism, the performance and energy efficiency of TSI-based memory systems were significantly improved compared to the conventional systems. In the μbank-based main memory system, simple open-page policy provides comparable performance to complex prediction-based page-management policies, thus simplifying the memory controller design. Future work includes further analysis on the impact of scheduling policies for the performance and energy efficiency of μbank memory systems.

6.1 Future Work

There are several interesting fields of future research based on the present work:

Integrating Simulation Framework for x86-based Accelerators

The modern high performance computer systems include both general purpose processors and accelerators to improve throughput [127]. Currently, McSimA+ does not simulate these systems due to the lack of accelerator modeling and Intel introduced widely used x86-based accelerator, Xeon Phi [128][129]. McSimA+ supports x86-ISA and implements its functional simulator using the Intel Pin binary instrumentation tool. Theoretically, the recently announced Xeon Phi architectures, such as Knight Corner [128] and Knight Landing [129] can be simulated because Pin [130] also supports binary instrumentation for an application running on Xeon Phi. The target simulation framework supporting a x86-based accelerator will contribute to the field of computer architecture as there is no simulation framework modeling systems that utilize x86 processors and x86 accelerators, despite their wide use.

μbank-aware Software

μbank can provide high performance and energy efficiency for emerging memory intensive applications (e.g. in-memory database [131][132] and in-memory map reduce [133][134]). Since a large amount of μbanks provides ample bank-level parallelism, some

can be dedicated to a specific program or a thread in a program to reduce inter-thread interference. This μbank-aware memory allocation can be done by modifying the OS memory allocator. Furthermore, the OS also provides a dedicated memory space for various data structures of its kernel like memory-mapped I/O, network stack, and others.

Design Space of Memory Controllers for μbank-based Main Memory Systems

In Chapter 5, we show that simple open page-management policy provides comparable performance and energy efficiency to the complex tournament page-management policy. A further review on the design space of modern memory controllers is needed, since they are generally designed by assuming a limited bank-level parallelism in memory systems. To implement an efficient memory controller for the μbank in terms of performance, energy, and cost, we should explore the design space of modern memory controllers, such as memory access scheduling policies, request buffer organizations, fairness of services.

Bibliography

[1] Y. H. Son, S. O, H. Yang, D. Jung, J. H. Ahn, J. Kim, J. Kim and J. W. Lee, "Microbank: Architecting Through-Silicon Interposer-Based Main Memory Systems," in *SC*, 2014.

[2] J. H. Ahn, S. Li, S. O and N. P. Jouppi, "McSimA+: A Manycore Simulator with Application-levle+ Simulation and Detailed Microarchitecture Modeling," in *ISPASS*, 2013.

[3] S. O, S. Choo and J. H. Ahn, "Exploring Energy-Efficient DRAM Array Organizations," in *MWSCAS*, 2011.

[4] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown and A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor," *Micro, IEEE,* vol. 27, no. 5, 2007.

[5] P. Greenhalgh, "Big.LITTLE Processing with ARM Cortex-A15 & Coretex-A7," *ARM White Paper,* 2011.

[6] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich and R. S. Schreiber, "Future Scaling of Processor-Memory Interfaces," in *SC*, 2009.

[7] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David and Z. Zhu,

"Mini-Rank: Adaptive DRAM Architecture for Improving Memory Power Efficiency," in *MICRO*, 2008.

[8] G. Gilbert, "Bringing In-Memory Transaction Processing to the Masses: an Analysis of Microsoft SQL Server 2014 in-memory OLTP," Microsoft, 2014.

[9] R. Ho, P. Amberg, E. Chang, P. Koka, J. Lexau, G. Li and F. Y. Liu, "Silicon Photonic Interconnects for Large- Scale Computer Systems," *Micro, IEEE,* vol. 33, no. 1, 2013.

[10] R.Weerasekera, J. R. Cubillo and G. Katti, "Analysis of Signal Integrity (SI) Robustness in Through-Silicon Interposer (TSI) Interconnects," in *Electronics Packaging Technology Conference*, 2012.

[11] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson and J. D. Owens, "Memory Access Scheduling," in *ISCA*, 2000.

[12] "International Technology Roadmap for Semiconductors," [Online]. Available: http://www.itrs.net/models.html.

[13] Micron, "Calculating Memory System Power for DDR3," 2007. [Online]. Available: http://www.micron.com/products/.

[14] U. Kang, H. J. Chung, S. Heo, S. H. Ahn, H. Lee, S. H. Cha, J. Ahn, D. M. Kwon, J. H. Kim, J. W. Lee, H. S. Joo, W. S. Kim, H. K. Kim, E. M. Lee, S. R. Kim, K. H. Ma and D. H. Jang, "8Gb 3D DDR3 DRAM using Through-Silicon-Via Technology," in *ISSCC*, 2009.

[15] J. S. Kim, C. S. Oh, H. Lee, D. Lee, H. R. Hwang, S. Hwang, B. Na, J. Moon, J. G. Kim, H. Park, J. W. Ryu, K. Park, S. K. Kang, S. Y. Kim and H. Kim, "A 1.2V 12.8GB/s 2Gb Mobile Wide-I/O DRAM with 4×128 I/Os Using TSV-Based Stacking," in *ISSCC*,

2010.

[16] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *MICRO*, 2010.

[17] S. O, Y. H. Son, N. S. Kim and J. H. Ahn, "Row-Buffer Decoupling: A Case for Low-Latency DRAM Microarchitecture," in *ISCA*, 2014.

[18] C. Kozyrakis, "Scalable Vector Media-processors for Embedded Systems," Ph.D. dissertation, University of California at Berkeley, 2002.

[19] Samsung, DDR3 SDRAM Datasheet, 2012.

[20] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," in *ISCA*, 2008.

[21] E. Ipek, O. Mutlu, J. F. Martínez and R. Caruana, "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," in *ISCA*, 2008.

[22] J. Mukundan and J. F. Martinez, "MORSE: Multi-objective Reconfigurable Selfoptimizing Memory Scheduler," in *HPCA*, 2012.

[23] D. Kaseridis, J. Stuecheli and L. K. John, "Minimalist Open-page: A DRAM Page-mode Scheduling Policy for the Many-core Era," in *MICRO*, 2011.

[24] Intel, "Intel Xeon Processor 7500 Series Datasheet," 2010.

[25] AMD, "BIOS and Kernel Developer's Guide (BKDG) for AMD Family," 2014.

[26] Y. Kim, M. Papamichael, O. Mutlu and M. Harchol-Balter, "Thread Cluster Memory Scheduling: Exploiting Differences in

Memory Access Behavior," in *MICRO*, 2010.

[27] Y. Kim, D. Han, O. Mutlu and M. Harchol-Balter, "ATLAS: A Scalable and High-performance Scheduling Algorithm for Multiple Memory Controllers," in *HPCA*, 2010.

[28] G. Chrysos, "Intel Many Integrated Core Architecture," in *Hot Chips*, 2012.

[29] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu and J. Hestness, "The GEM5 Simulator," *Computer Architecture News,* vol. 39, no. 2, 2012.

[30] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill and D. A. Wood, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset," *Computer Architecture News,* vol. 33, no. 4, 2005.

[31] A. Patel, F. Afram and K. Ghose, "MARSSx86: Micro Architectural Systems Simulator," in *ISCA Tutorial Session*, 2012.

[32] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi and J. C. Hoe, "SimFlex: Statistical Sampling of Computer System," *Micro, IEEE,* vol. 26, no. 4, 2006.

[33] M. T. Yourst, "PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator," in *ISPASS*, 2007.

[34] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep and A. Agarwal, "Graphite: A Distributed Parallel Simulator for Multicores," in *HPCA*, 2010.

[35] P. M. Ortego and P. Sack, "SESC: SuperESCalar Simulator," UUIC, Tech. Rep, 2004.

[36] T. E. Carlson, W. Heirman and L. Eeckhout, "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation," in *SC*, 2011.

[37] T. Austin, E. Larson and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *Computer,* vol. 35, no. 2, 2002.

[38] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim and W. J. Dally, "A Detailed and Flexible Cycle-Accurate Network-on- Chip Simulator," in *ISPASS*, 2013.

[39] N. Agarwal, T. Krishna, L. S. Peh and N. K. Jha, "GARNET: A Detailed On-Chip Network Model Inside a Full-System Simulator," in *ISPASS*, 2009.

[40] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong and T. M. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *ISPASS*, 2009.

[41] P. Rosenfeld, E. Cooper-Balis and B. Jacob, "DRAMSim2: a Cycle Accurate Memory System Simulator," *Computer Architecture Letters,* vol. 10, no. 1, 2011.

[42] J. Edler and M. D. Hill, "Dinero IV," [Online]. Available: http://www.cs.wisc.edu/~markhill/DineroIV.

[43] G. H. Loh, S. Subramaniam and Y. Xie, "Zesto: A Cycle-Level Simulator for Highly Detailed Microarchitecture Exploration," in *ISPASS*, 2009.

[44] A. Jaleel, R. S. Cohn, C. Luk and B. Jacob, "Cmp$im: A Binary Instrumentation Approach to Modeling Memory Behavior of Workloads on CMPs," University of Maryland, Tech. Rep, 2006.

[45] J. Moses, K. Aisopos, A. Jaleel, R. Lyer, R. Illikkal, D. Newell and S. Makineni, "CMPSched$im: Evaluating OS/CMP interaction on shared cache management," in *ISPASS*, 2009.

[46] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt and B. Werner, "Simics: A Full System Simulation Platform," *Computer,* vol. 35, no. 2, 2002.

[47] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *ATEC*, 2005.

[48] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi and K. Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," in *PLDI*, 2005.

[49] M. Monchiero, J. H. Ahn, A. Falcón, D. Ortega and P. Faraboschi, "How to Simulate 1000 Cores," *Computer Architecture News,* vol. 37, no. 2, 2009.

[50] H. Pan, K. Asanovi´c, R. Cohn and C. K. Luk, "Controlling Program Execution through Binary Instrumentation," *Computer Architecture News,* vol. 33, no. 5, 2005.

[51] D. R. Butenhof, Programming with POSIX threads, 1997.

[52] R. Kumar and G. Hinton, "A Family of 45nm IA Processors," in *ISSCC*, 2009.

[53] P. Kongetira, K. Aingaran and K. Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor," *Micro, IEEE,* vol. 25, no. 2, 2005.

[54] S. Li, J. H. Ahn, R. D. Strong, J. B. B. D. M. Tullsen and N. P. Jouppi, "McPAT: an Integrated Power, Area, and Timing

Modeling Framework for Multicore and Manycore Architectures," in *Micro*, 2009.

[55] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel and A. Adileh, "Scale-Out Processors," in *ISCA*, 2012.

[56] J. T. Pawlowski, "Hybrid Memory Cube (HMC)," in *Hot Chips*, 2011.

[57] Intel, "P6 Family of Processors Hardware Developer's Manual," *Intel White Paper,* 1998.

[58] M. Hassan, L. Warriner, K. Yen, B. Upputuri, D. Greenhill, A. Kumar and H. Park, "An 8-core 64-thread 64b power-efficient SPARC SoC," in *ISSCC*, 2007.

[59] A. Jain, W. Anderson, T. Benninghoff, D. Berucci, M. Braganza, J. Burnetie and T. Chang, "A 1.2 GHz Alpha Microprocessor with 44.8 GB/s Chip Pin Bandwidth," in *ISSCC*, 2001.

[60] A. Gupta, W. D. Weber and T. Mowry, "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes," in *ICPP*, 1990.

[61] J. L. Baer and W. H. Wang, "On the Inclusion Properties for Multi-Level Cache Hierarchies," in *ISCA*, 1988.

[62] B. W. O'Krafka and A. R. Newton, "An Empirical Evaluation of Two Memory-Efficient Directory Methods," in *ISCA*, 1990.

[63] D. Abts, N. D. E. Jerger, J. Kim, D. Gibson and M. H. Lipasti, "Achieving Predictable Performance through Better Memory Controller Placement in Many-Core CMPs," in *ISCA*, 2009.

[64] D. Burger, S. Hily, S. Mckee, P. Ranganathan and T. Wenisch, "Cycle- Accurate Simulators: Knowing When to Say When," in *ISCA Panel Session*, 2008.

[65] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta, "The SPLASH-2 programs: Characterization and Methodological Considerations," in *ISCA*, 1995.

[66] Intel, "Intel VTune Performance Analyzer," [Online]. Available: http://software.intel.com/en-us/intel-vtune.

[67] Intel. [Online]. Available: http://www.intel.com/content/www/us/en/processors/atom/atom.

[68] N. Hardavellas, M. Ferdman, B. Falsafi and A. Ailamaki, "Reactive NUCA: Near-Optimal Block Placement and Replication in Distributed Caches," in *ISCA*, 2009.

[69] J. L. Henning, "SPEC CPU2006 Memory Footprint," *Computer Architecture News,* vol. 35, no. 1, 2007.

[70] V. Craeynest, Kenzo, A. Jaleel, L. Eeckhout, P. Narvaez and J. Emer, "Scheduling Heterogeneous Multi-cores Through Performance Impact Estimation (PIE)," in *ISCA*, 2012.

[71] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis and N. P. Jouppi, "Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores," in *ISCA*, 2010.

[72] S. Beamer, C. Sun, Y. J. Kwon, A. Joshi, C. Batten, V. Stojanović and K. Asanović, "Re-Architecting DRAM Memory Systems with Monolithically Integrated Silicon Photonics," in *ISCA*, 2010.

[73] "CACTI6.5," [Online]. Available: http://www.hpl.hp.com/research/cacti.

[74] "SPEC CPU2006," [Online]. Available: http://www.spec.org/cpu2006.

[75] J. L. Hennessy and D. A. Patterson, Computer Architecture: A

Quantitative Approach, 5th ed, Morgan Kaufmann Publishers Inc, 2012.

[76] S. Damaraju, V. George, S. Jahagirdar, T. Khondker, R. Milstrey, S. Sarkar, S. Siers, I. Stolero and A. Subbiah, "A 22nm IA multi-CPU and GPU System-on-Chip," in *ISSCC*, 2012.

[77] C. Johnson, D. Allen, J. Brown, S. Vanderwiel, R. Hoover, H. Achilles, C.-Y. Cher, G. May, H. Franke, J. Xenedis and C. Basso, "A Wire- Speed PowerTM processor: 2.3GHz 45nm SOI with 16 cores and 64 threads," in *ISSCC*, 2010.

[78] V. Krishnaswamy, D. Huang, S. Turullols and J. L. Shin, "Bandwidth and Power Management of Glueless 8-Socket SPARC T5 System," in *ISSCC*, 2013.

[79] Cooper-Balis, P. R. Elliott and B. Jacob, "Buffer-On-Board Memory System," in *ISCA*, 2012.

[80] J. R. Cubillo, R. Weerasekera, Z. Z. Oo, E.-X. Liu, B. Conn, S. Bhattacharya and R. Patti, "Interconnect Design and Analysis for Through Silicon Interposers (TSIs)," in *3DIC*, 2012.

[81] S. Lakka, "Xilinx SSI Technology, Concept to Silicon Development Overview," in *Hot Chips Tutorial*, 2012.

[82] J. Standard, "GDDR5 SGRAM Datasheet," 2013.

[83] J. Standard, "Low Power Double Data Rate 2 (LPDDR2)," 2010.

[84] W. J. Dally and J. W. Poulton, Digital Systems Engineering, Cambridge University Press, 1998.

[85] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman and N. P. Jouppi, "CACTI-3DD: Architecture-level Modeling for 3D Die-stacked DRAM Main Memory," in *DATE*, 2012.

[86] W. Zhao and Y. Cao, "New Generation of Predictive Technology

Model for Sub-45nm Design Exploration," in *ISQED*, 2006.

[87] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung and S. Hong, "25.2 A 1.2V 8Gb 8- channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV," in *ISSCC*, 2014.

[88] Y. H. Son, S. O, J. W. L. Yuhawn Ro and J. H. Ahn, "Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations," in *ISCA*, 2013.

[89] D. H. Yoon, J. Chang, N. Muralimanohar and P. Ranganathan, "BOOM: Enabling Mobile Memory Based Low-Power Server DIMMs," in *ISCA*, 2012.

[90] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel 그리고 M. Horowitz, "Energy-performance Tradeoffs in Processor Architecture and Circuit Design: a Marginal Cost Analysis," %1 *ISCA*, 2010.

[91] T. Singh, J. Bell and S. Southard, "Jaguar: A Next-Generation Lowpower x86-64 Core," in *ISSCC*, 2013.

[92] D. W. Chang, Y. H. Son, J. H. Ahn, H. Kim, M. Ahn, M. J. Schulte and N. S. Kim, "Dynamic Bandwidth Scaling for Embedded DSPs with 3D-stacked DRAM and Wide I/Os," in *ICCAD*, 2013.

[93] B. Jacob, S. W. Ng and D. T. Wang, Memory Systems: Cache, DRAM, Disk, Morgan Kaufmann Publishers Inc, 2007.

[94] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian and O. Mutlu, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.

[95] Y. Kim, V. Seshadri, D. Lee, J. Liu and O. Mutlu, "A Case for

Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.

[96] E. Ebrahimi, R. Miftakhutdinov, C. Fallin, C. J. Lee, J. A. Joao, O. Mutlu and Y. N. Patt, "Parallel Application Memory Scheduling," in *Micro*, 2011.

[97] C. Bienia, S. Kumar, J. P. Singh and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *PACT*, 2008.

[98] "TPC Benchmark," [Online]. Available: http://www.tpc.org.

[99] T. Sherwood, E. Perelman, G. Hamerly and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in *ASPLOS*, 2002.

[100] "PostgreSQL," [Online]. Available: http://www.postgresql.org.

[101] J. H. Huang, "Opening Keynote: Pascal Next-Generation GPU," in *GPU Technology Conference*, 2014.

[102] S. S. Iyer, "The Evolution of Dense Embedded Memory in High Performance Logic Technologies," in *IEDM*, 2012.

[103] D. Jevdjic, S. Volos and B. Falsafi, "Die-Stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache," in *ISCA*, 2013.

[104] G. H. Loh and M. D. Hill, "Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches," in *MICRO*, 2011.

[105] N. Madan, N. M. L. Zhao, A. Udipi, R. Balasubramonian, R. Iyer, S. Makineni and a. D. Newell, "Optimizing Communication and Capacity in a 3D Stacked Reconfigurable Cache Hierarchy," in *HPCA*, 2009.

[106] N. D. Gulur, R. Manikantan, M. Mehendale and R. Govindarajan, "Multiple Sub-Row Buffers in DRAM: Unlocking Performance and Energy Improvement Opportunities," in *ICS*, 2012.

[107] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang and Y. Xie, "Half-DRAM: a High-bandwidth and Low-power DRAM Architecture from the Rethinking of Fine-grained Activation," in *ISCA*, 2014.

[108] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi and R. Balasubramonian, "Micro-pages: increasing DRAM efficiency with localityaware data placement," in *ASPLOS*, 2010.

[109] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen and O. Mutulu, "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems," in *HPCA*, 2013.

[110] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets and B. Verghese, "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," in *ISCA*, 2000.

[111] J. L. Henning, "Performance Counters and Development of SPEC CPU2006," *Computer Arhicteucture News,* vol. 35, no. 1, 2007.

[112] K. K. Rangan, G.-Y. Wei and D. Brooks, "Thread motion: Fine-grained power management for multi-core systems," in *ISCA*, 2009.

[113] K. Hu, R. Bai, T. Jiang, C. Ma, A. Ragab, S. Palermo and P. Y. Chiang, "0.16-0.25 pJ/bit, 8 Gb/s Near-Threshold Serial Link Receiver With Super-Harmonic Injection-Locking," *JSSC,* vol. 47, no. 8, 2012.

[114] J. Standard, "High Bandwidth Memory DRAM Datasheet," 2013.

[115] H.-W. Lee, S.-B. Lim, J. Song, J.-B. Koo, D.-H. Kwon, J.-H.

Kang, Y. Kim, Y.-J. Choi, K. Park, B.-T. Chung and C. Kim, "A 283.2 uW 800Mb/s/pin DLL-based data self-aligner for Through-Silicon Via (TSV) interface," in *ISSCC*, 2012.

[116] N. Miura, K. Kasuga, M. Saito and T. Kuroda, "An 8Tb/s 1pJ/b 0.8mm2/Tb/s QDR Inductive-Coupling Interface Between 65nm CMOS GPU and 0.1um DRAM," in *ISSCC*, 2010.

[117] Y. Ohara, A. Noriki, K. Sakuma, K.-W. Lee, M. Murugesan, J. Bea, F. Yamada, T. Fukushima, T. Tanaka and M. Koyanagi, "10 um Fine Pitch Cu/Sn Micro-Bumps for 3-D Super-Chip Stack," in *3DIC*, 2009.

[118] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie, "Half-DRAM: a High-bandwidth and Low-power DRAM Architecture from the Rethinking of Fine-grained Activation," in I*SCA*, 2014.

[119] J. Shao and B. T. Davis, "A Burst Scheduling Access Reordering Mechanism," in *HPCA*, 2007.

[120] A. Silberschtz and P. B. Galvin, and G. Gagne, "Operating System Concepts 8th edition," 2008.

[121] J. Chow, B. Pfaff, T. Garfinkel, and M. Rosenblum, "Shredding Your Garbage: Reducing Data lifetime Through Secure Deaalocation,", in *USENIX Security*, 2008.

[122] A. M. Dunn, M. Z. Lee, S. Jana, S. Kim, and M. Silberstein, "Eternal Sunshine of the Spotless Machine: Protecting Privacy with Ephemeral Channels,", in *OSDI*, 2011.

[123] C. A. Waldspurger, "Memory Resource Management in VMware ESX Server,", in *OSDI*, 2002.

[124] M. Vrable, J. Ma, J. Chen, D Moore, E. Vandekieft, A. C. Snoeren, Geoffrey, M. Voelker, and S. Savage "Scalability,

Fidelity, and Containment in the Potemkin Virtual Honeyfarm,", *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5, 2005.

[125] "STREAM Benchmark," [Online]. Available: https://www.cs.virginia.edu/stream/.

[126] Micron, LPDDR2 SDRAM Datasheet, 2010.

[127] J. Shalf, S. Dosanjh, and J. Morrison, "Exascale Computing Technology Challenges,", in *VECPAR*, 2010.

[128] G. Chrysos and Senior Principle Enginner, "In tel Xeon Phi Coprocessor (Codename Knights Corner,", in *Hot Chips*, 2012.

[129] G. Chrysos, "Intel Xeon Phi Coprocessor-the Architecture," *Intel White Paper,* 2014.

[130] Intel, "Pin-A Dynmaic Binary Instrumentation Tool," [Online]. Available: https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool.

[131] F. Farber, S. K. Cha, J. Primsch, C. Bornhovd, S. Sigg, and W. Lehner, "SAP HANA Database: Data Management For Modern Business Applicaions,", *ACM Sigmod REcord*, vol. 40, no. 4, 2012.

[132] H. Lim, D. Han, D. G. Anderson, and M. kaminsky, "MICA: A Holistic Approach to Fast In-memory Key-value Storage,", in *NSDI*, 2014.

[133] J. Talbot, R. M. Yoo, and C. Kozyrakis, "Phoenix++: Momular MapReduce for Shared-memry Systems,", in *MapReduce Workshop*, 2011.

[134] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Scott, and S. Ion, "Spark: Cluster Computing with Working Sets,", in *USENIX Hot Topics*, 2010.

국문초록

최근 매니코어 프로세서 기반 시스템은 학계 및 업계의 주목을 받고 있으며, DRAM 은 최신 매니코어 시스템의 주기억장치로 널리 사용되고 있다. 응용프로그램 또한 매니코어 프로세서를 효과적으로 활용하기 위해 병렬화되고 있으며, 이들이 처리하는 데이터양이 증가함에따라 컴퓨터 시스템의 주기억장치 용량은 계속 증가하고 있다. 따라서, 주기억장치 시스템은 최신 매니코어 프로세서 시스템의 성능 및 에너지효율 결정하는 중요한 요소로 작용한다. 최근 주목 받는 TSI (Through-Silicon Interposer) 기술은 매니코어 시스템을 위한 고대역폭, 고에너지효율 특성을 갖는 주기억장치 시스템의 구성을 가능하게 한다. TSI 기반 주기억장치 시스템은 I/O 에너지효율을 향상 시키는 반면, DRAM 코어에 의해 소비되는 에너지가 컴퓨터 시스템에서 소비되는 전체 에너지 소비 특성을 결정하는 “불균형”적인 주기억장치 시스템 설계를 초래한다. 따라서 주기억장치로 널리 활용되는 DRAM 장치 내부구조가 매니코어 시스템의 성능 및 에너지효율에 미치는 영향에 대한 연구가 필요하다.

최신 매니코어 시스템 관련 연구를 위해서는 코어 및 언코어 하위장치들의 내부구조가 정교하게 구현된 매니코어 시뮬레이터가 필요하다. 매니코어 프로세서는 복잡한 코어 및 언코어 하위장치들을 하나의 칩 (Chip)에 집적한 SoC (System-on-Chip)이다. 코어 하위 장치들은 전통적인 대칭 코어나 비대칭 코어로 구성된다. 언코어 하위장치들은 정교한 대용량 다계층 캐시와, 칩 내부 장치 연결을 위한 고성능 상호연결망, 다수의 주기억장치 제어기 및 주기억장치 (DRAM)으로 구성되며, 이들의 복잡도는 날로 증가하고 있다.

연구 목표인 매니코어 시스템을 위한 효율적인 주기억장치 시스템의 구성을 위해, 새로운 매니코어 시뮬레이터인 McSimA+를 개발하였다. McSimA+는 코어 및 언코어 하위장치의 내부구조가 정교하게 구현된 cycle-level 매니코어 시뮬레이터이다. McSimA+는 application-level+ 시뮬레이터로 기존 full-system 시뮬레이터와 application-level 시뮬레이터의 장점을 포괄한다. 따라서 본 연구에서 개발한 McSimA+는 application-level 시뮬레이터의 장점인 빠른 시뮬레이션 속도를 유지하면서 full-system 시뮬레이터와 같은 쓰레드 및 프로세스 제어가 가능하다.

본 연구에서는 개발한 매니코어 시뮬레이터를 이용하여, DRAM 장치 내부구조가 매니코어 시스템 의 성능 및 에너지효율에 미치는 영향을 분석하였다. 이를 위해, 우선 널리 사용되는 메모리 모델링 도구인 CACTI 를 개조하여 최신 DRAM 의 뱅크 수와 뱅크의 페이지 크기를 변경하면서 DRAM 장치 내부구조에 대한 설계공간을 탐구하였다. 모델링 결과 DRAM 뱅크 (bank)의 페이지 (DRAM row) 크기가 클수록 면적효율과 지연시간이 향상되는 반면, activation/precharge 에너지는 늘어남을 보였다. 모델링 결과를 바탕으로 3 차원 집적된 주기억장치를 포함하는 매니코어 시스템에서 DRAM 장치 내부구조가 시스템의 성능 및 에너지효율에 미치는 영향을 탐구하였다. 뱅크 수가 늘어남에 따라 시스템 성능 및 에너지효율은 증가하며, 8KB 페이지가 최적의 성능을 보이지만, 2KB 페이지에서 최적의 에너지효율을 얻을 수 있음을 확인하였다.

본 논문에서는 앞선 연구 결과에서 얻은 DRAM 장치 내부구조 및 매니코어 시스템에 대한 이해를 바탕으로, 매니코어 시스템을 위한 TSI 기반 주기억장치 시스템에서 발생하는 에너지 소비의 “불균형” 문제를 해결할 수 있는 새로운 TSI 기반 주기억장치 시스템을 개발하였다.

새로운 TSI 기반 주기억장치 시스템은 본 연구에서 제안하는 고성능 DRAM 장치 내부구조인 마이크로뱅크를 활용한다. 마이크로뱅크는 5% 이하의 적은 면적 증가로 기존 DRAM 뱅크를 다수의 독립적으로 작동 가능한 작은 뱅크로 분할한다. 따라서, 마이크로뱅크는 주기억장치 시스템의 뱅크 수준의 병렬성을 향상시켜 TSI 기반 주기억장치 시스템의 성능 및 에너지효율을 개선한다. 마이크로뱅크 기반 주기억장치 시스템을 활용 하면 주기억장치 시스템의 성능저하를 일으키는 bank conflict 발생 빈도가 감소하므로, 단순한 open-page policy 로도 복잡한 prediction-based page-management policy 와 동등한 성능을 얻을 수 있다.

McSimA+를 이용한 실험결과 마이크로뱅크 기반 주기억장치 시스템이 적용된 매니코어 시스템에서 open-page policy 와 이상적인 page-management policy 의 성능차이는 평균 5% 이하였다. 따라서 마크로뱅크를 적용한 주기억장치 시스템은 복잡한 page-management policy 의 구현을 요구하지 않으므로, 주기억장치 시스템의 설계를 단순화할 수 있다. 우리의 마이크로뱅크 기반 주기억장치 시스템이 적용된 매니코어 시스템과 기존 DDR3 DRAM 기반 주기억장치가 적용된 매니코어 시스템에서, 빈번하게 메모리 접근을 수행하는 SPEC CPU2006 응용프로그램들의 성능 및 에너지효율을 비교하였다. 실험결과 마이크로뱅크 기반 주기억장치 시스템이 적용된 매니코어 시스템에서 이들 워크로드에 대해 평균적으로 1.62 배의 성능과 4.8 배의 에너지효율의 향상을 얻을 수 있음을 확인하였다.

주요어 : 매니코어, 주기억장치 시스템, DRAM, 마이크로뱅크, 매니코어 시뮬레이터, Through-Silicon Interposer (TSI)
학 번: 2009-23807