ABSTRACT

Title of thesis:	PERFORMANCE STUDY OF VARIOUS MODERN DRAM ARCHITECTURES
	Dhiraj Reddy Nallapa Yoge Master of Science, 2018
Thesis directed by:	Professor Bruce Jacob Department of Electrical and Computer Engineering

Several DRAM architectures exist with each differing in their performance, power and cost metrics. This thesis compares the performance and power characteristics of some of such DRAM architectures which are compliant to JEDEC standard DDR protocols such as DDR3, DDR4, LPDDR3, LPDDR4, GDDR5 and HBM. To accurately model the differences in performance and power characteristics of these architectures, a new cycle level DRAM memory simulator has been designed and implemented from scratch. Several distinguishing features of these protocols such as - bankgroups in DDR4 and beyond, 32 activation window constraint in GDDR5, granularity of refresh at per rank level vs at per bank level and dual command issue mode in HBM - are modeled and studied for their impact on workload performance and power consumption. The internal structure of DRAM exhibits different kinds of parallelisms such as channel level parallelism, rank level parallelism and bank level parallelism. The type and the degree of parallelism together with the associated DRAM command timing constraints determine the latency and bandwidth characteristics of any DRAM architecture. Abstract studies are performed to determine the potential of each of these parallelisms in attaining the maximum supported pin bandwidth for a set of SPEC 2006 CPU workloads. Finally, several real DRAM architecture designs belonging to each of the above mentioned protocols are studied to quantify their relative performance and power trade-off.

PERFORMANCE STUDY OF VARIOUS MODERN DRAM ARCHITECTURES

by

Dhiraj Reddy Nallapa Yoge

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2018

Acknowledgments

I would like to thank my advisor professor Bruce Jacob for providing the opportunity to work under his guidance. I would like to particularly thank him for always granting the freedom to work on topics and projects that interested me. I would also like to thank professors Donald Yeung and Rajeev Barua for agreeing to be part of my thesis committee. Also, a special thanks to Melanie Prange for helping me in getting all the paperwork correct during my final semester. Above all I would like to thank my family and friends for always being there for me and making all this worthwhile.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

DRAM	Dynamic Random Access Memory
JEDEC	Joint Electron Device Engineering Council
DDR	Double Data Rate
HBM	High Bandwidth Memory
SPEC	Standard Performance Evaluation Corporation
LLC	Last Level Cache
DIMM	Dual Inline Memory Module
t_{RAS}	Row Access Strobe Latency
t_{RCD}	Row to Column Delay
t_{RTRS}	Rank to Rank turn around delay
t_{CCD}	Column to Column delay between banks
t_{CCDS}	Column to Column delay between banks of different bankgroups
t_{CCDL}	Column to Column delay between banks of the same bankgroup
t_{RRD}	Row to Row Activation delay
t_{FAW}	Four Activation Window
t_{32AW}	Thirty two Activation Window
t_{RTP}	Read to Precharge delay
t_{RFC}	Refresh Cycle
t_{WTR}	Write to Read turnaround time
SST	Structural Simulation Toolkit
MPKI	Misses Per Kilo Instructions
IPC	Instructions Per Cycle

Chapter 1: Introduction

With the ever increasing core counts in modern day processors the demands on the memory subsystem are only increasing. For all but the very simplest of applications, the characteristics of the memory subsystem play a critical role in determining the overall application performance. With the end of Denard's scaling, the clock frequencies of processor cores have not increased markedly over the past decade. However, the number of cores on a single processor have been increasing continuously [1]. Along with the increase in the number of cores, the ever exploding working set sizes of modern day applications have increased the bandwidth requirement to service all last level cache (LLC) misses.

While there is a very active research on several alternative device technologies as the building block for the main memory, still most of the commercial computers of the day use Dynamic Random Access Memory (DRAM) as the underlying technology of choice for building the main memory subsystem. While DRAM memory capacity has mostly kept pace with the increase in core counts in processors, neither the access latency nor the memory bandwidth have increased at the same rate.

The various DRAM protocol timing parameters which determine the average access latency have at best stayed constant if not worsened slightly with each successive DRAM generations. This is because most of the slack gained with moving to newer process nodes has been largely utilized to increase the number of bits that could be packed on a DRAM device of a given silicon area. As a result, the access latencies even for an unstressed memory subsystem haven't improved notably.

For off package memories, the raw memory bandwidth that is available to a processor depends on the number of memory channels the processor can support and the memory bandwidth that a Dual Inline Memory Module (DIMM) in each channel can provide. The number of physical pins on the periphery of a CPU die is a critical resource and the number that can be dedicated for the interconnection with the memory subsystem limits the number of true memory channels that a processor can have. While the pin bandwidth that a DIMM can provide has increased from DDR3 [2] to DDR4 [3], it has nowhere kept pace with the rate of increase in core counts in processors. As a result of limited number of channels and limited bandwidth per channel, the burden of providing sufficient bandwidth consummate with the number of cores has become a first class constraint in processor design. On package memories such as High Bandwidth Memory (HBM) [4] have been proposed as a solution to increase the raw bandwidth. Since the HBM memory die lies on the same silicon substrate as the processor die, the number of wires connecting the processor die and the memory die is no longer limited by the number of peripheral physical pins of the CPU. As a result, HBM can have much larger number of channels thereby supporting a higher bandwidth. However, while moving the memory die on package somewhat solves the bandwidth problem by increasing the number of channels, it does cost heavily in terms of memory capacity and hence overall memory cost. As a result, most of the commercial processors only use HBM as either a hardware managed DRAM cache or as a faster DRAM as part of a heterogeneous memory system [5], [6].

While memory subsystem design is critically important for high performance computing, it is no less important for low-power mobile computing. The memory subsystem consumes a non-trivial amount of total system power, and there is demand for DRAM architectures that provide similar or slightly degraded performance at much lower power. The LPDDR [7], [8] class of memories is designed specifically with these objectives in mind.

The DRAM manufacturing vendors are typically separate from the CPU vendors. Hence, to facilitate interoperability between the the CPU's memory controller and DRAM modules, the DRAM interface has been standardized by organizations such as JEDEC. JEDEC standardizes the interface by describing a DRAM protocol in terms of the various commands the DRAM module understands and the timing constraints that need to be obeyed between the issuance of these commands for reliable operation. The various JEDEC standardized protocols studied in this thesis are DDR3 [2], DDR4 [3], LPDDR3 [7], LPDDR4 [8], GDDR5 [9] and HBM [4]. While the DRAM protocol only mentions the commands and timing constraints, the actual DRAM architectures that obey a given protocol could be widely different in terms of their capacity and internal organization.

In this thesis, some of such DRAM architectures obeying the various DRAM protocols are studied for their relative power and performance trade-offs. To facilitate such a study, a comprehensive cycle-level DRAM simulator has been built from scratch. The simulator models the behaviour of a close to optimal memory controller for each of these protocols. This simulator is used to study the various salient features of the DDR protocols. Finally, studies are conducted to quantity the performance differences among several DRAM architectures belonging to different DDR protocols.

1.1 Organization of the thesis

This thesis is organized as follows. Chapter 2 briefly describes the internal organization of DRAM and throws light on the various timing constraints. Chapter 3 shows the design of a cycle level DRAM simulator illustrating the various software design aspects involved in building a modular architectural simulator. Chapter 4 studies the various distinguishing features of some of the newer DDR protocols and showcases the impact of these features on performance and power. Chapter 5 performs a comparative study of the various DRAM architectures and reports their relative power performance trade-offs. Chapter 6 concludes this thesis by summarizing the various key aspects of the work.

Chapter 2: DRAM Overview

This chapter gives a brief overview of the internal organization of DRAM and the typical steps involved in performing a memory access. It also explains how the various protocol-specific timing parameters impact the application performance.

2.1 Internal Organization and Basic Operation

Each data cell in a DRAM consists of a capacitor and an access transistor. A set of such data cells arranged in rows and columns forms a memory array. Figure 2.1 shows a stylized representation of a memory array and a data cell. A column of a DRAM row can be 4-bit (X4), 8-bit (X8) or 16-bit (X16) wide depending upon the number of memory arrays that act as a single unit. Further, the data-width of a column determines the number of DRAM devices that make up a Bank of a Rank. For example, for DRAM devices with 4-bit (X4), 8-bit (X8) or 16-bit (X16) column data-width, the number of devices that make up the 64-bit wide data bus is 16, 8 or 4 respectively. Fig 2.2 shows a stylized representation of a bank made up of four X16 DRAM devices. On a memory access, a column data-width number of bits is transferred from each DRAM device in a clock tick. A 64 byte data access occurs over 8 ticks (4-cycles in a double-data-rate DRAM), with 64-bits being transferred

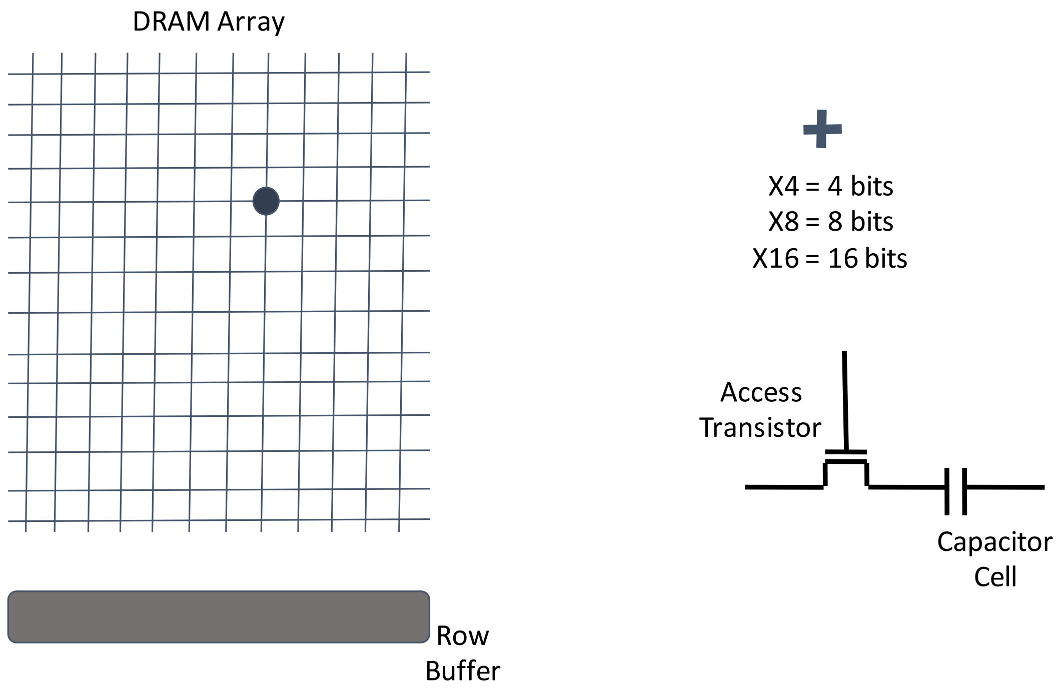

Figure 2.1: Stylized representation of a DRAM array

on each tick. The 64-bits that are transferred in a tick are spread across the DRAM devices constituting a rank, with each device delivering a column-width number of bits.

A typical memory access cycle on a bank of a DRAM involves the steps of *ACTIVATE*, *READ/WRITE* and *PRECHARGE* [10]. Initially, assuming that the column bit-lines and the sense amplifiers of the row buffer are in the Precharged state, the row from which the memory access is to be performed is Activated. During Activation, the contents of an entire row spread across all the devices that form a bank of a rank are so read into the sense amplifiers of the row-buffer. This process is destructive in nature in the sense that immediately after Activation the charge stored in the capacitor cells of the activated row is no longer valid. The activation process is

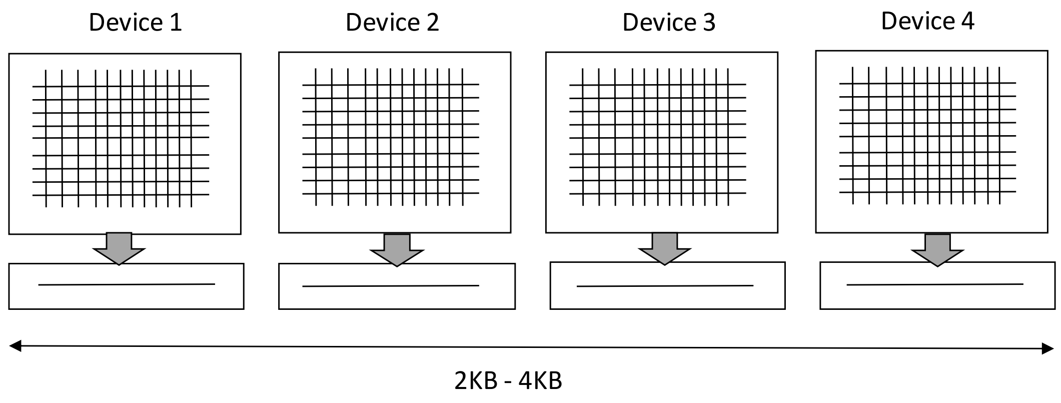

Figure 2.2: Stylized representation of a bank consisting for 4 X16 DRAM devices

illustrated in the Figure 2.3. It will take a time of t_{RAS} (Row Access Strobe Latency) for the charge in the capacitor cells of the activated row to be restored to their original values. However, after only a time of t_{RCD} (row to column delay) from the activation, typically much smaller than t_{RAS}, the data stored in the sense-amplifiers of the row buffer is available to be read or written. The stylized representation of the read/write operation is shown in Figure 2.4. Once the read/write operations finish and the charge in the sense amplifiers is restored to the capacitor cells of the memory array's row, the column bit-lines are precharged to make them ready to activate a different row if required. Multiple read/write operations can be performed on the data stored in the sense amplifiers before precharge. Since the size of the row buffer (2KB - 4KB) is typically quite large compared to a single memory access (64B), the process of activating an entire row becomes prohibitively expensive if it is required to be done for every memory access. Hence, the entire DRAM internal architecture and memory controller design is centered around performing as many read/write requests as possible on an activated row before the bit-lines are Precharged. An

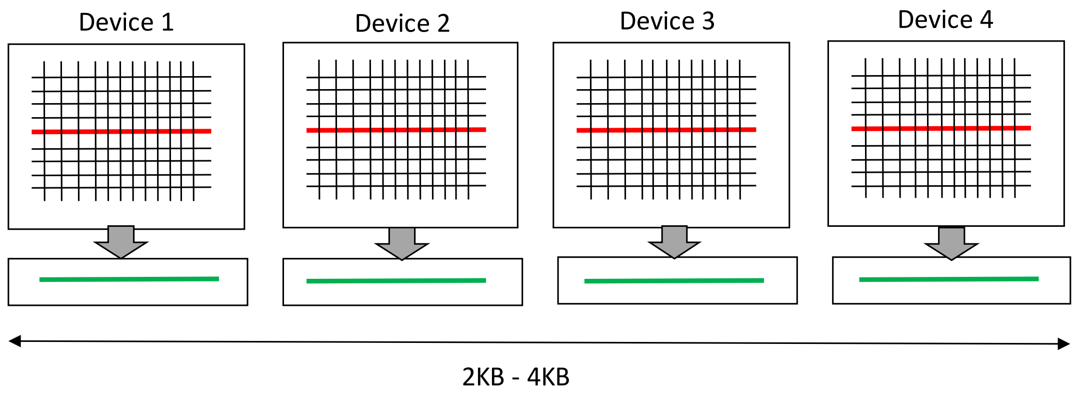

Figure 2.3: Stylized representation of Activation process in a bank of DRAM

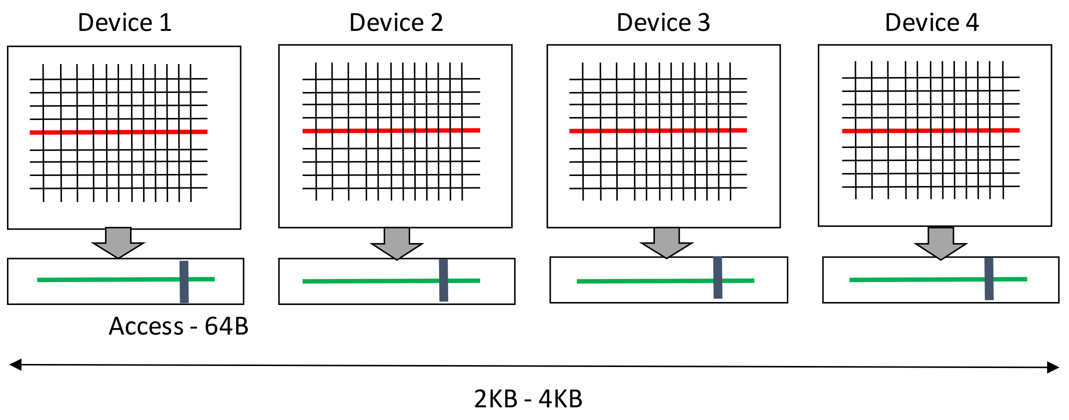

Figure 2.4: Stylized representation of read/write process in a bank of DRAM

activated row is also called an open row and a memory access performed on an open row is called a row buffer hit.

In addition to the typical Activate - Read/write - Precharge cycle, the leaky nature of the charge on the capacitor cells requires another operation to be performed periodically. The charge in the capacitor cells of a set of DRAM rows is restored periodically by an operation called as Refresh. During refresh, the banks on which the refresh is being performed are unavailable for any memory accesses. As a result,

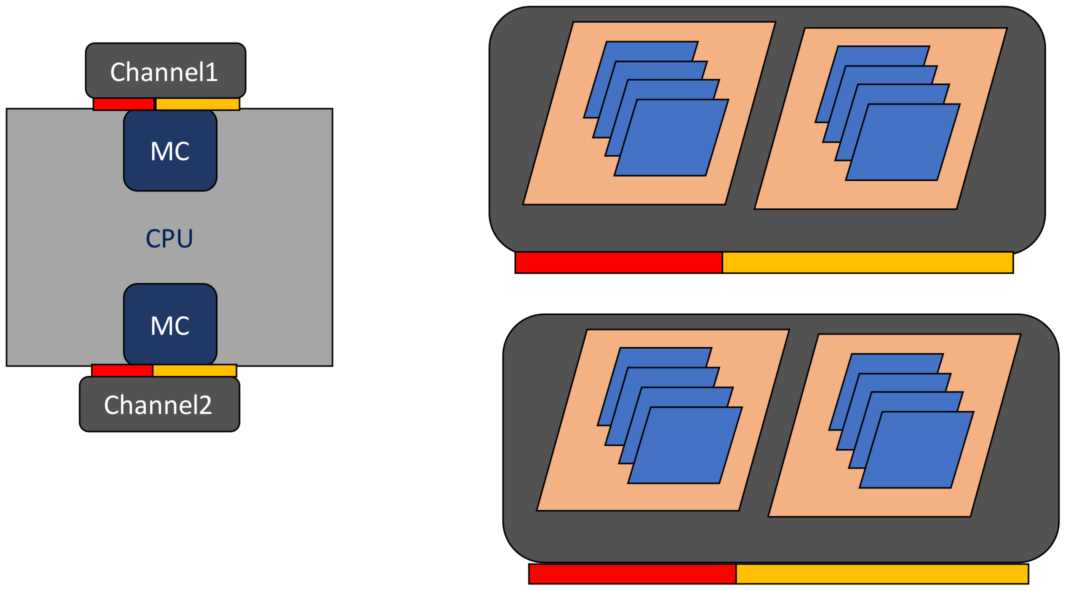

Figure 2.5: Stylized representation of a 2 channel memory system with 2 ranks per channel and 4 banks per rank

refresh operations can significantly increase the latency of those memory accesses that arrive during the refresh process to the banks being refreshed. The performance penalty of refresh can be reduced by utilizing the slack in when the Refresh operation needs to be strictly performed as per the DDR protocol specification.

Several DRAM banks are grouped together to form a rank, and multiple ranks can be present in a single Channel of a DRAM. Fig 2.5 shows a stylized representation of a CPU connected to a two channel memory system with two ranks per channel and 4 banks per rank. Each memory channel has its own memory controller responsible for performing memory accesses for the addresses mapped to that channel. A channel has its own command and data buses and hence increasing the number of channels increases both the amount of memory parallelism available as well as the maximum available memory bandwidth. On the other hand, the ranks of a channel and the banks within a rank all share the same command and data bus. Hence, while having a greater number of banks per channel doesn't increase the maximum available memory bandwidth, it does increase the number of memory accesses that can pipelined for data transfer across the data bus. As we will investigate in this thesis, the amount of internal parallelism available in DRAM is a critical parameter in determining the fraction of maximum available bandwidth that a memory sub-system can actually deliver.

2.2 Timing Constraints

The DDR protocols specify the various commands that need to be issued to perform the basic operations of activation, read, write, precharge, refresh etc. In addition to specifying the necessary command sequences, the DDR protocols also specify various timing constraints that need to be obeyed between the issuance of the DRAM commands. These timing constraints are an indirect way of expressing the various electrical constraints that manifest during the manufacturing process as well those that arise due to sharing of internal structures. Some of these timing constraints exist to express the time delay for the various operations to happen while others exist to limit the peak current. The four activation window constraint is one such example which exists to limit the peak current profile as activation operation draws a relatively high amount of current. Also, a command issued to a certain part of the DIMM could create a timing dependencies for the issuance of commands to other parts of the DIMM. This section briefly discusses some of these timing constraints and how they directly impact memory access latency and thereby workload performance.

All the ranks of a channel share the same command and data buses. The sharing of the data bus imposes the obvious timing constraint that a memory access from a rank has to wait till the ongoing memory access from a different bank finishes. In addition, an extra wait time of t_{RTRS} (Rank-to-Rank turn around delay) is required when switching the usage of the data bus to a different rank. Additional timing constrains apply when the two ranks perform different types of accesses i.e. when the one rank does a read access and the other rank has to do a write access or vice versa. Hence, frequent switching between accesses to different ranks as well frequently switching between read and write accesses could cause a access latency penalty.

The t_{CCD} (Column-to-Column delay) timing constraint imposes the minimum time delay required between the issuance of two column access commands to different banks of a rank. When t_{CCD} is greater than the time required to transfer the data across the data bus (*tBurst*), it becomes the limiting factor in determining the effective utilization of bank-level parallelism. In DDR4 and beyond architectures the set of banks in a rank are grouped into two or more groups called as bankgroups. For these architectures, the t_{CCD} timing constraint splits into two parameters namely - t_{CCDS} (t_{CCD} small) and t_{CCDL} (t_{CCD} large), wherein the column access commands to banks of the same group require a larger delay between them than those to banks belonging to different bankgroups. As a result, greater latency penalty would be paid if the successive accesses are not interleaved well across banks belonging to different bankgroups.

The t_{RRD} (Row-to-Row activation delay) timing constraint imposes the minimum delay required between the issuance of two activate command to different banks of a rank. In addition, the t_{FAW} (Four-activation-window) timing constraint specifies a restriction on the number of activate commands that can issued to banks of a rank in any contiguous time interval of length t_{FAW}. The t_{FAW} is typically larger than four times t_{RRD}. These two constraints together determine how quickly the memory accesses belonging to different banks of a rank can be performed thereby determining how well the bank level parallelism can be utilized.

In addition to the above mentioned constraints, numerous other timing constrains exist which together determine the performance characteristics of the memory subsystem. The complete details of these timing constraints can be found in the JEDEC protocol specification data sheets. It is paramount for the memory controller to take into consideration the relative values of the various timing constraints to decide on the optimal memory request scheduling algorithms.

Chapter 3: DRAM Simulator Design and Simulation Methodology

Simulation is an important aspect of computer architecture research. It is seldom practical to build actual hardware to evaluate an architectural idea. On the other hand, analytical models are often too simplistic and end up not giving a real insight. Simulation provides a fine balance between the accuracy at which an idea can be evaluated versus the cost of evaluating an idea in terms of both time and money. Hence, building and using simulators is the preferred approach for doing performance evaluation and exploration studies. This chapter describes the design of a new DRAM memory simulator as well as the overall simulation methodology used to perform the various studies in the rest of thesis

Architectural simulators come in numerous forms with varying degrees of accuracy, modularity and simulation speed. Often, sacrificing some amount of accuracy is unavoidable to make the simulation time tractable. There are several popular architectural simulators such as gem5 [11], Zsim [12], sniper [13], Structural Simulation toolkit (SST) [14] etc. Each of these simulators has its own internal model of the DRAM memory controller. However, the modeling of the memory controller is often not fully accurate. While such inaccuracy might not significantly affect the kind of studies for which these simulator were designed, it could produce misleading results for certain kinds of memory performance studies. For example, the very design of simulators such as Sniper and Zsim makes it impossible to model the effect of back-pressure due to memory controller queues being full. While it is conventional wisdom that some of these inaccuracies can be safely ignored, depending upon the type of study, doing so might not always be the right thing to do. For example, a study to quantify the effect of limited memory bandwidth on overall performance would be measured incorrectly if simulators such as Zsim or Sniper are used. This is because one of the first order effects of having a limited bandwidth is to cause the processor to stall due to the queues in the memory controller being full, a behavior which doesn't get modelled while using Zsim or Sniper.

While there are a few existing publicly available DRAM memory simulators such as DRAMSim2 [15], USIMM or Ramulator [16] which model some of the features of some of the protocols, there are none which either model or could be easily modified to model all the features of interest in our study. The DRAMSim2 and USIMM simulators were primarily designed to model the features of DDR2 and DDR3 protocols. While Ramulator is close to our simulator in its capability to model various newer DRAM protocols the software design of Ramulator requires the various protocol specific timing parameters to be compile time constants which restricts the ease of doing certain kinds of performance studies.

Stand alone cycle level DRAM simulators such as DRAMSim2 [15], Ramulator [16] etc model the memory controller behavior much more accurately albeit being a lot slower than some of the internal DRAM controller models available in popular architectural simulators. In this thesis, a new cycle level DRAM simulator is designed and implemented to accurately model the behavior of various DDR protocols.

3.1 Design of a new DRAM simulator

A DRAM simulator essentially models the behavior of a memory controller performing read and write accesses to the memory subsystem. The memory controller issues the requisite commands to the DRAM devices while obeying the various timing constraints of the corresponding DDR protocol. In addition, the key duty of a high performance memory controller is to orchestrate the scheduling of memory accesses so as to maximize performance or minimize power. Design of efficient memory access scheduling algorithms is a very active area of research. All commercial processors employ proprietary memory scheduling algorithms, the details of which are not public. In this DRAM simulator, a memory scheduling algorithm which is generally accepted to be close to optimal has been implemented.

The key aspects in the software design of a DRAM simulator are as below -

1. To maintain the correct state information across all the parts of the DIMMs at all times.

2. To maintain timing constraints depicting the earliest time at which a particular DRAM command can be issued to different parts of a DIMM.

3. To design a high performance memory scheduling algorithm that serves memory requests out-of-order to maximize row buffer hits. At the same time, the

scheduling algorithm has to maintain some degree of fairness to avoid starvation. In addition, the memory controller has to periodically refresh the cells in the DRAM devices to maintain data fidelity. Issuing refreshes while causing minimal performance loss is another important design constraint.

Each of these design aspects are described in further detail in the following sub-sections

3.1.1 Maintaining internal state

Some of the state information that is maintained in the simulator is as below

1. *Is Bank Open* - Whether a row buffer is open in a bank or not and if it is open which row is open. This state information is maintained for each bank.

2. *In Self Refresh* - Whether a DIMM is in self refresh mode or not. This state information is maintained for each channel(DIMM).

3. *Last Four Activations* - The clock cycles at which the last 4 activation commands were issued. This state information is maintained for each rank and is used to the obey the t_{FAW} timing constraint.

4. *Last 32 Activations* - The clock cycles at which the last 32 activation commands were issued. This state information is maintained for each rank and is used to obey the t_{FAW} timing constraint in GDDR5.

A crucial aspect of the simulator design is to always maintain the correct state information. So, whenever any DRAM command is issued, it is checked to see if it modifies any of the above state and if so, the corresponding state is updated appropriately. The following examples illustrates how some of these state updates occur:

1. Suppose an activate command is issued to a bank with no open row. In this case, the states *Is Bank Open*, *Last Four Activations* and *Last 32 Activations* are updated. *Is Bank Open* state is updated as the activate command causes a row in a bank to open. *Last Four Activations* and *Last 32 Activations* states are updated as the new activate command counts as one among the last few activate commands.

2. Suppose a precharge command is issued to a bank with an open row. In this case, only the state *Is Bank Open* is updated as the precharge will cause the open row in the bank to close.

3. A column read or write command does not change any state information.

The current state of the different parts of the DIMM determines if any command needs to be issued as a prerequisite before issuing the command to perform the actual read/write operations. The following examples illustrates how the current state is queried to determine if any prerequisite command is required to be issued, and if so the corresponding prerequisite command is returned.

1. If a Read command is to be issued to bank in which a different row is open, the state information of the bank is queried to determine that the required prerequisite command is the precharge command.

2. If a Read command is to be issued to a DIMM which is in currently in self-refresh mode, the state information is queried to determine that the required prerequisite command is the self-refresh exit command.

3. If a Read command is to be issued to bank in which the command is addressed to the same row as the currently open row (i.e.a row buffer hit occurs) then the state information is queried to determine that no prerequisite command needs to be issued and the issuance of the Read command can go ahead.

Thus, the correct current state information is maintained by updating the state after the issuance of any state changing command. Also, the state information is queried to determine the corresponding prerequisite command, if any, that needs to issued before the desired command for performing read/write accesses.

3.1.2 Maintaining timing constraints

The various DRAM protocol timing constraints are captured by storing, for every DRAM command, the earliest time at which it can be issued to any part of the DIMM. Accurately maintaining this earliest time information is the key design aspect that makes the DRAM simulator functionally correct.

Whenever a command is issued to a part of the DIMM, it creates a timing dependency for the subsequent issuance of a set of commands to different parts of the DIMM. The details of such dependencies are what makes the DRAM protocols so complex. This timing dependency between the issuance of a command and the subsequent issuance of a set of commands is stored as a static data structure whose values are populated once during the initialization phase of the DRAM protocol in the simulator. This data structure captures all the subtle details of the protocol timing constraints in a clean manner.

On the issuance of a DRAM command, the earliest time information for all the dependent subsequent commands to different parts of the DIMM are updated. The following examples illustrates how some of this timing information is updated in the simulator:

1. Suppose a *Read* command is issued to one of the banks of a rank. This creates several timing dependencies such as - subsequent *Read* commands cannot be issued to the banks of the same rank before a time delay of t_{CCD} (column-to-column delay), a *Precharge* command cannot be issued to this bank before a time delay of t_{RTP} (read-to-precharge delay) etc. So, on the issuance of the original *Read* command, the earliest time information is updated for the *Read* commands to the same rank and the *Precharge* command to the same bank.

2. Suppose a *Refresh* command is issued to a bank of a DIMM. This creates a timing dependency that an *Activate* command cannot be issued to the same bank before the time delay of t_{RFC} (refresh cycle delay). So, the earliest time information for the *Activate* command to this bank is updated to be not before t_{RFC}.

3. Suppose a *Write* command is issued to a rank of a DIMM. This creates a timing dependency that *Read* commands even to different ranks cannot be issued before a delay of t_{WTR} (write-to-read turnaround delay) after the data transfer

finishes ($tBURST$). So, the earliest time information for write commands to different ranks is updated to be not before ($t_{BURST} + t_{WTR}$).

The memory controller queries the earliest time information before the issuance of any command so as the ensure protocol timing compliance. A command is issued only if the current clock tick of the simulator is past the earliest time at which that command could be issued. Checking the earliest time information before the issuance of any DRAM command and updating it appropriately for all dependent commands after the issuance is how the timing protocol constraints are accurately met in the simulator.

3.1.3 Memory controller model and memory access scheduling

The key duties of the memory controller are to read the data stored in the DIMM, to write data to the DIMM, and to maintain the integrity of the data in the DIMM. To perform these functions the memory controller sends various commands to the DIMMs through the command bus. It is expected that memory controller sends these commands while observing the specific DRAM protocol requirements. The behavior of the DRAM devices when the memory controller fails to observe the protocol specifications is undefined and could lead to corruption of the data stored on the DRAM devices.

A high performance memory controller also needs to schedule the memory accesses to minimize average latency and increase overall memory throughput. There's a wide array of research on several kinds of memory scheduling algorithms with some prioritizing the overall throughput, whereas others try to give greater importance to ensuring fairness among memory accesses. The optimal memory scheduling in a given scenario is highly dependent upon the characteristics of the workloads being run.

Each channel of the memory system has an independent memory controller that is responsible for performing the memory accesses to that channel. The memory scheduling algorithm of such a memory controller that has been implemented in the built simulator is briefly described below. The memory controller has queues to store the incoming memory access requests from the CPU. These queues could be one per channel or one per rank or even one per each bank. A memory request queue is searched to determine if a DRAM command useful to service a memory request in the queue could be issued in this cycle. If no such command is found, the remaining queues are searched in a round robin manner. The algorithm tries to service the memory requests mostly in a first-come-first-serve order, and only looks to service the memory requests out-of-order when potential row buffer hits would be missed due to strictly adhering to the first-come-first-serve policy. In addition, to avoid possible starvation if out-of-order memory requests resulting in row buffer hits are forever prioritized, a hard limit on the number of row buffer hits for prioritization is set. After reaching the set limit, the out-of-order row buffer hits are no longer prioritized, and the required command for servicing the oldest request in the queue is prioritized.

In addition to performing the read/write memory requests from the CPU, the memory controller has to periodically issue refresh commands to restore the stored charge in the leaky DRAM capacitor cells. In the simulator, this is done by placing Refresh requests in a separate refresh-request queue. The requests in the refresh queue could either all be rank-level refresh requests or all bank-level refresh requests or a combination of both. The memory-scheduling algorithm in the simulator prioritizes issuing refreshes over new read/write memory requests when there are pending refresh requests in the refresh queues. So, when the refresh queue is non-empty, i.e. a refresh request is pending, the memory controller does not issue any new read/write requests to the part of the DIMM that is awaiting refresh and waits for existing accesses to complete before issuing the required command to perform refresh.

The state information, the earliest time information and the memory access scheduling algorithm together form the building blocks of the DRAM simulator. On each cycle, the memory controller queries the state and timing information to decide on the memory access to be scheduled and issues the corresponding DRAM command for it. After a DRAM command is issued, the state information and earliest time information are updated. Finally, when a memory access request finishes, it is returned to the CPU simulator, which gets to correctly model the memory access latency for that memory request.

3.1.4 Support for all DDR protocols

The simulator has been designed with generality and flexibility in mind. Instead of designing a separate memory controller and separate state and timing information storage for each of the DDR protocols, a single module which supports all the features of the various protocols has been implemented with the ability to enable or disable the various features as required.

Each of new features of the DRAM protocols require modifications to the memory controller's memory-access scheduling algorithm to take their full advantage. The details of some of such features and how the memory-access scheduling algorithm has been modified to accommodate them is described below.

Self-refresh modes exist in some low power DDR architectures. During the self-refresh mode the memory controller doesn't have to issue any Refresh commands, and refresh operation is managed internally by the DRAM DIMM itself [17]. When to enter and when to exit the self-refresh mode is a decision made by the memory controller. In the simulator, a self-refresh command is issued when the memory controller queues are found to be empty for a certain number of cycles. That is, if no memory requests were sent from the CPU for over a period of time, in the expectation that memory requests might not be sent even for some more time in the future, the memory controller asks the DRAM DIMM to enter self-refresh mode. Some additional state information required for this is whether or not all queues are empty, and the clock cycle from which the queues have been empty. The self-refresh exit is done when a read/write request gets queued in the memory controller. While it is occasionally possible to speculatively exit the self-refresh mode preemptively, for the sake of simplicity, it is not done in the simulator's memory-controller implementation.

In HBM architectures, two DRAM commands can be issued in each cycle. The two commands that can be issued in the same cycle cannot be arbitrary, but they need to belong to different "groups". One group is of row commands consisting of *Activate* and *Precharge* commands. The other group is of column commands consisting of *Read* and *Write* commands. To utilize the potential of dual command issue in improving performance, the memory controller is modified to look for issuing two such complimentary commands each cycle.

Some of the newer architectures have the feature to issue refreshes at both rank and bank granularities. To study the usefulness of having the ability to issue bank level refresh, several different refresh strategies have been implemented in the simulator.

With a memory controller that utilizes the newer features of the different protocols and follows timing specifications of different DDR protocols, DRAM architectures belonging to different protocols can be compared using the built simulator.

3.1.5 Integration with CPU simulators

There two primary approaches to performing memory architectural studies using a DRAM simulator. One approach is to collect the memory traces for workloads by either running them directly on a real machine or by running them only once on a detailed CPU simulator and collecting the memory address traces. These memory address traces are then used as representative workloads for performance evaluation using the memory simulator. This approach is still widely popular because of its simplicity and often times being the only way to do a tractable study especially for systems with large numbers of processor cores. However, such studies could end up being inaccurate and often give false insights because the key aspect of the feedback from the memory sub-system causing the processor to stall and not be able to issue any more memory requests won't get modelled.

The other approach is to integrate the DRAM simulator with a CPU simulator by substituting its memory controller model with the controller model in the DRAM simulator. The designed memory simulator has been integrated with several front-end CPU simulators such as structural simulation toolkit(SST), Zsim and Gem5. Some of the issues which were encountered while integrating with the front-end simulators are describe below.

Zsim [12] is a fast CPU simulator, but its software design doesn't allow the modeling of back-pressure to the CPU due to queues in the memory controller being full. We found this to be a source of inaccuracy for doing performance comparison of dram architectures with widely differing bandwidth characteristics. Both Structural simulation toolkit(SST) [14] and gem5 [11] are capable of modeling the back-pressure to the CPU due to the memory request queues being full. However, the only CPU model available in SST, called Ariel, is a fairly simplistic model of an in-order CPU with IPC = 1 for non-memory instructions. We found that this simple in-order CPU model was incapable of creating the necessary memory bandwidth demand required to evaluate the benefits of different DRAM architectures. Eventually, an out-of-order CPU model in gem5 was chosen as the right CPU model to go along with our detailed cycle level DRAM simulator.

3.2 Simulation Methodology

This section describes the simulation methodology used to perform the various studies in this thesis. The configuration of the CPU simulator, memory controller as well as the characteristics of the various studied workloads are described in the following subsections.

3.2.1 CPU Simulator

To keep the simulation time tractable, a methodology of obtaining workload checkpoints using a fast functional simulation and then doing the actual detailed simulation by restoring from these checkpoints has been employed. First, each workload is run using the gem5 AtomicSimpleCPU model and check-pointed after running for 10 Billion instructions per core. The obtained single-core checkpoints are combined to form a multi-core multi-process checkpoint. The detailed simulations are done by restoring from the multi-core process checkpoint and employing a detailed Out-of-order CPU model for the benchmark execution after restoration from the checkpoint. Multi-core simulations are performed by executing the SPEC workloads in rate mode. The configuration of the CPU simulator used to perform the various studies is show in Table 3.1. The common memory controller configuration that is used for performing the various studies is shown in 3.2

Table 3.1: Configuration of the CPU simulator gem5

CPU	Gem5 OOO CPU model, x86 architecture, 8-cores Syscall emulation mode (SE mode)
Core	4GHz, Out-of-order, 8-fetch, 8-issue, 192 reorder buffer entries
L1 I-Cache	per-core, 32KB, 2-way associative, 64 Byte cache line, LRU
L1 D-Cache	per-core, 64KB, 2-way associative, 64 Byte cache line, LRU
L2 Cache	private, 2MB, 8-way associative, 64 Byte cache line, LRU
Workloads	bzip2, mcf, milc, leslie3d, soplex, GemsFDTD, lbm, astar, sphinx3
Checkpointing	Multicore process checkpoints at 10 Billion instructions
Number of Instructions	100 million instructions per core A total of 800 million instructions

3.2.2 Workloads studied and their characterization

A selection of SPEC CPU2006 workloads with high memory bandwidth requirements is used to conduct the various studies. All the SPEC CPU2006 workloads were characterized for their last-level-cache (LLC) misses behaviors. A selection of these workloads with high misses-per-kilo-instructions (MPKI) is used for performing various studies. The MPKI characterization of these workloads is show in Figure 3.1

Table 3.2: Common configuration of the memory controller that is used for various studies

Address Mapping	robabgrachco row-bank-bankgroup-rank-channel-column
Queue Structure	Per Bank Queues
Queue Depth	16
Refresh Strategy	Rank-level Staggered
Request Scheduling	First-come-first-serve while prioritizing row buffer hits

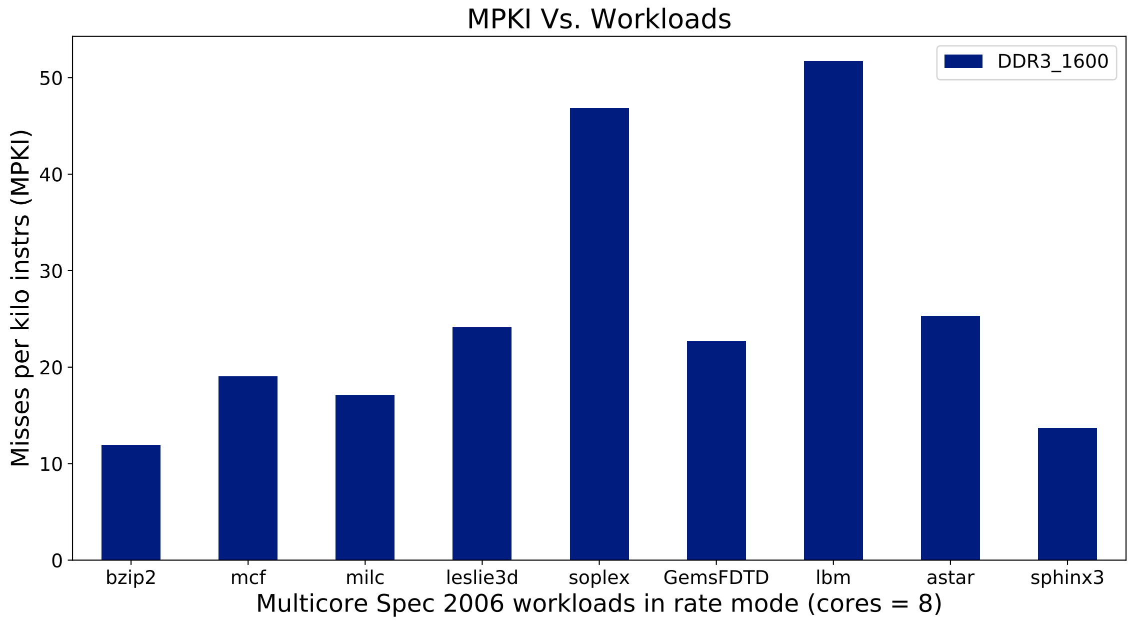

Figure 3.1: Misses-per-kilo-instructions (MPKI) characterization of workloads

Chapter 4: Salient features of different protocols

In this chapter the various distinguishing features of different DDR protocols are studied for their impact on performance and power.

4.1 Bankgroups in DDR4 and beyond

DRAMs exhibit bank-level parallelism i.e. read/write accesses to the open rows in different banks of a rank can be serviced in an interleaved manner. In contrast, a bank conflict is said to occur if memory accesses mapped to different rows of the same bank need to be serviced in an interleaved manner. Increase in the number of bank conflicts decreases the number of possible row buffer hits due to spatial locality and thereby severely degrades application performance and power characteristics because of increased activation and precharge overhead. Therefore, having a large number of banks per rank is a highly desired DRAM architecture feature.

The minimum time that is needed between the issuance of two successive read command or two successive write commands to the banks of a rank is the maximum of the following two parameters $tBURST$ and t_{CCD}. Figure 4.1 shows the DRAM timing diagram with t_{CCDS} and t_{CCDL} parameters.

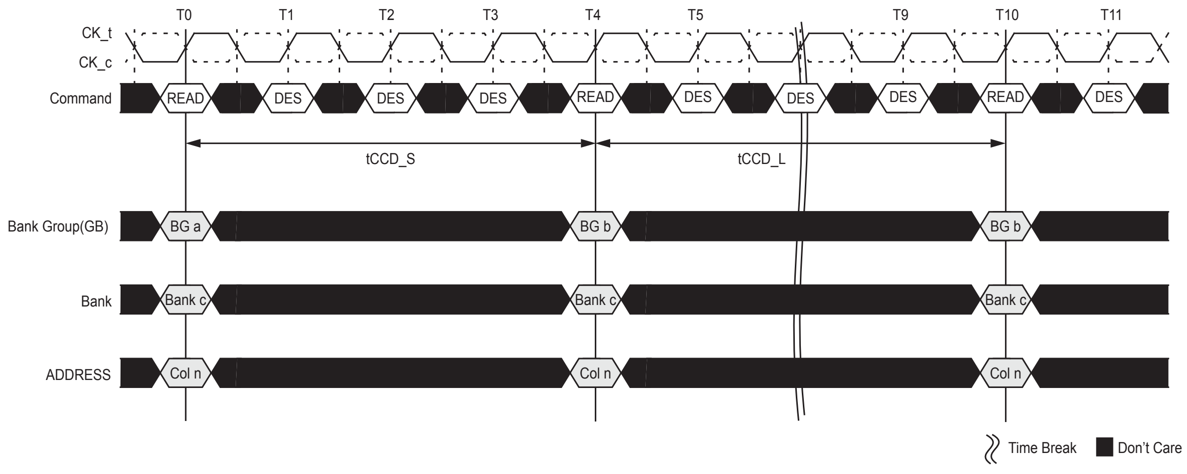

Figure 4.1: t_{CCD} Timing

1. *tBURST* - The time required to transfer the data across the peripheral pins which is determined by the burst length of the DRAM.

2. t_{CCD} - The column-to-column command timing constraint which determines the minimum required time interval between the issuance of two column access commands to the banks of a rank.

DDR4 and beyond architectures such as GDDR5, HBM, LPDDR4 introduce the concept of bankgroups. Instead of treating all the banks of a rank alike, the banks of a rank are divided into two or more bankgroups. The protocol timing constraint parameter t_{CCD} is now split into two timing parameters - t_{CCDL} (t_{CCD} large) and t_{CCDS} (t_{CCD} small). t_{CCDL} is the minimum required time interval between the issuance of two column commands to banks belonging to the same bankgroup, whereas t_{CCDS} is the minimum required time interval between the issuance of two column commands to the banks belonging to different bankgroups. t_{CCDL} is greater than

t_{CCDS} and therefore successive accesses to banks belonging to different bankgroups are less costly than successive accesses to banks belonging to the same bankgroup. The concept of bankgroups can be seen as a practical compromise to increase the number of possible banks per rank without paying the penalty for the associated worst case increase in t_{CCD}. To increase the total number of banks while obeying the physical design constraints, banks are grouped together into bankgroups where in the accesses to banks in the same bankgroup have different timing constraints to obey than to those in different bankgroups.

The concept of bankgroups helps increase the bank level parallelism by increasing the total number of available banks per rank in a DRAM architecture. However, if the successive accesses are not interleaved well across banks belonging to different bankgroups, there is a greater latency penalty to pay than before. If the accesses are interleaved well across bankgroups, the overall performance would be closer to a DRAM architecture without bankgroups and t_{CCD} equal to t_{CCDS}. On the other hand, if the accesses are not interleaved well across bankgroups the overall performance would be closer to an architecture without bankgroups and t_{CCDS} equal to t_{CCDL}. Simulation studies are performed to quantify the impact of bankgroups on workload performance.

4.1.1 Bankgroups Vs Without bankgroups

A DRAM architecture consisting of 1 channel, 2 ranks and 8 banks is studied for how the division of banks into bankgroups affects performance. The 8 banks per

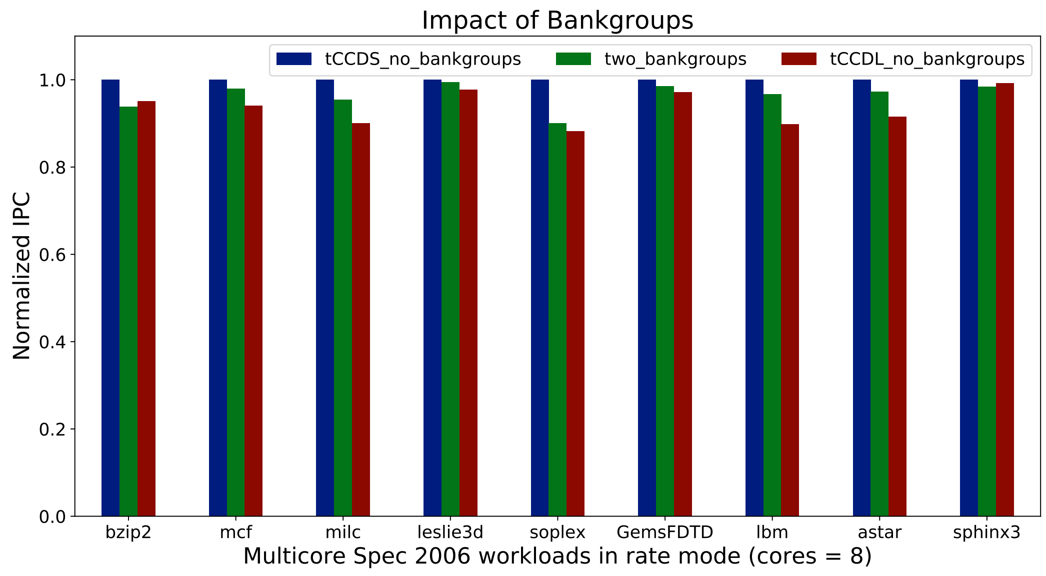

Figure 4.2: Impact of bankgroups

rank are divided into 2 bankgroups (i.e. 4 banks per bankgroup) with timing parameters t_{CCDL} and t_{CCDS}. These are compared with DRAM architectures without bankgroups and having 8 banks per rank with timing parameter $t_{CCD} = t_{CCDS}$ and $t_{CCD} = t_{CCDL}$ respectively. While it is expected that performance of the DRAM architecture with 2 bankgroups would be bounded on either side by those without bankgroups and with $t_{CCD} = t_{CCDS}$ and $t_{CCD} = t_{CCDL}$ respectively, the degree to which its performance is worse than the architecture without bankgroups and $t_{CCD} = t_{CCDS}$ showcases the performance penalty of having bankgroups.

Figure 4.2 plots the normalized IPC versus different bankgroup timing parameters for the various workloads. On average, having bankgroups and having different timing parameters t_{CCDS} and t_{CCDL} for accesses to within the same bankgroup and to different bankgroups causes a normalized IPC performance drop of 4% when compared to a DDR3 like structure with all banks with t_{CCDS} timing. However, increasing the worst case t_{CCD} delay to the t_{CCDL} value for all banks without the bankgroups feature would result in a much higher normalized IPC performance drop of as much as 6%. So, it appears that the bankgroups feature serves as a good compromise between the need to increase the total number of banks per rank without severely degrading the worst case column-to-column delay.

4.1.2 Varying the number of bankgroups

For a DRAM architecture with 1 channel, 1 rank and 32 banks per rank, the number of bankgroups into which these banks are sub-divided is varied from 1 to 32 at multiples of powers of 2 (i.e. 1, 2, 4, 8, 16, 32). It is expected that an increase in the number of bankgroups improves performance since the number of banks per bankgroup decreases and so does the probability of successive accesses belonging to the banks of the same bankgroup. The cases with bankgroups equal to 1 and 32 are essentially same as having no bankgroups and t_{CCD} equal to t_{CCDL} and t_{CCDS} respectively.

Figure 4.3 plots the normalized IPC versus different number of bankgroups for the various workloads. The plot demonstrates the an average normalized IPC gain of 12% is obtained for the case of 2 bankgroups with t_{CCDS} and t_{CCDL} parameters when compared to a single bankgroup with t_{CCDL}. However, the gains for higher numbers of bankgroups is only marginally more than the two bankgroups case.

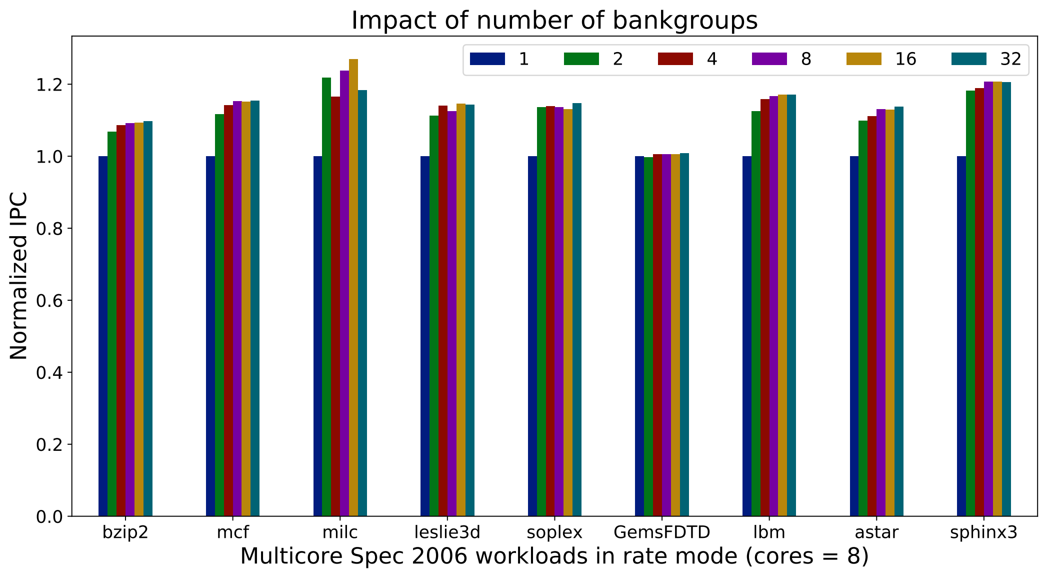

Figure 4.3: Varying number of bankgroups

4.2 32 activation window timing constraint in GDDR5

It is well known that the four activation window timing constraint (t_{FAW}) in DDR architectures is an important timing parameter. It limits the number of row activations that can be performed on banks of a rank in a time interval of t_{FAW}. Up to a maximum of 4 row activation commands can be issued in a running time interval of t_{FAW}. This timing constraint primarily exists to limit the maximum current profile of DRAM devices, as row activations are quite intensive in the current drawn. In addition to t_{FAW} timing constraint, GDDR5 DRAM architectures also have a 32-bank-activation window timing constraint (t_{FAW}), which analogously limits the number of row activations that can be done in a running time interval of t_{FAW} to a maximum of 32.

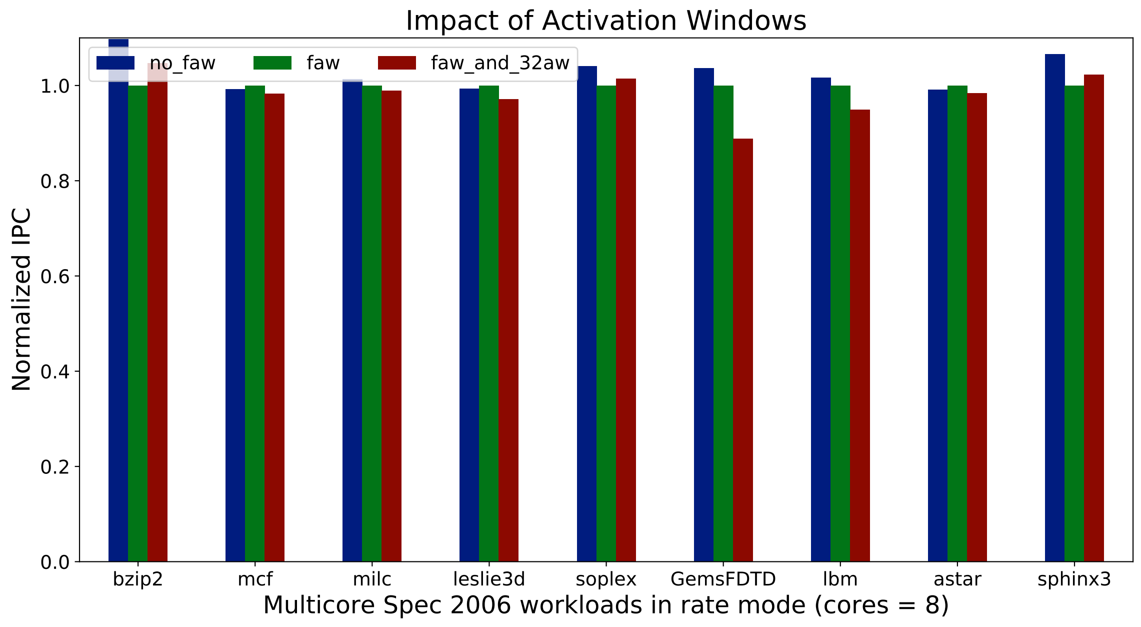

Figure 4.4: Impact of activation window constraints

Simulation studies are performed to quantify to what extent the 32-bank-activation-window timing constraint affects the performance of GDDR5 DRAM architectures compared to not having such a timing constraint. Since, the precise values of t_{FAW} are not specified in the GDDDR5 specifications, a t_{FAW} period equal to 10 times the t_{FAW} period is used to perform this study. Figure 4.4 plots the normalized IPC for the three cases - first, with no activation window constraints whatsoever not even t_{FAW}, second, with the baseline four window activation constraint and third with both 4-window and 32-window activation constraints. The IPC's are normalized w.r.t to baseline case of having only the t_{FAW} timing constraint. We observe that the 32-activation window constraint causes a normalized IPC drop of about 2% while not having the four window activation constraint itself would cause a normalized IPC performance increase of about 3%

4.3 Refresh granularity - Rank level Vs. Bank level

Newer DRAM architectures have the ability to issue refresh commands at the granularity of a single bank instead of issuing a refresh command for the entire rank at once. When a bank or a rank is getting refreshed, it is unavailable for servicing any memory requests. As a result, if refresh is done at a large granularity i.e. for the entire rank at once, a greater chunk of physical memory address space would be unavailable at the same time for servicing any memory requests [18] [19]. This could result in some performance degradation for latency-sensitive applications. On the other hand, performing a refresh at a smaller granularity, i.e. separately for each bank, would require a larger number of refresh commands be issued, thereby possibly causing some bottleneck at the command bus. In addition, providing the facility to issue per-bank refresh requires a greater amount of internal storage for bookkeeping to keep track of the next row to be refreshed for each bank separately, thereby marginally increasing the cost.

Four refresh-command-issuing strategies are compared for their impact on workload performance. The details of the refresh strategies are described below:

1. *RANK_LEVEL_SIMULTANEOUS* - Refresh commands are issued at a per rank granularity, and refresh commands to all the ranks of a channel are queued at the same time and issued one after the other. Essentially, in this strategy the entire channel is unavailable during the period of refresh.

2. RANK_LEVEL_STAGGERED - Refresh commands are issued at a per rank

granularity but refresh commands to different ranks of a channel are issued in a staggered manner periodically once every tREFI/number of ranks. This ensures that only a single rank is blocked due to refresh at a time.

3. BANK_LEVEL_SIMULTANEOUS - Refresh commands are issued at a per bank granularity, and refresh commands to all banks of a rank are queued at the same time and issued one after the other. Essentially, in this strategy, the entire rank is unavailable during the period of refresh in spite of issuing refreshes at per bank granularity.

4. BANK_LEVEL_STAGGERED - Refresh commands are issued at a per bank granularity, but refresh commands to different banks of a rank are issued in a staggered manner periodically every tREFIb/number of banks. This ensures that only a single bank of rank is unavailable due to refresh at any given time.

While RANK_LEVEL_SIMULTANEOUS and BANK_LEVEL_SIMULTANEOUS refresh strategies look sub-optimal for refresh at rank granularity and bank granularity respectively, depending upon the phases of memory access behaviour in workloads, they could be just as good if not better than the staggered refresh strategies.

Simulation studies are performed to quantify the relative merit of each of these refresh strategies on workload performance. Figure 4.5 shows the normalized IPC for different refresh strategies. In our simulations, some of the workloads benefit with issuing of refreshes at bank level whereas some others don't. On average, issuing refreshes at bank level causes a normalized IPC performance improvement of about 3%.

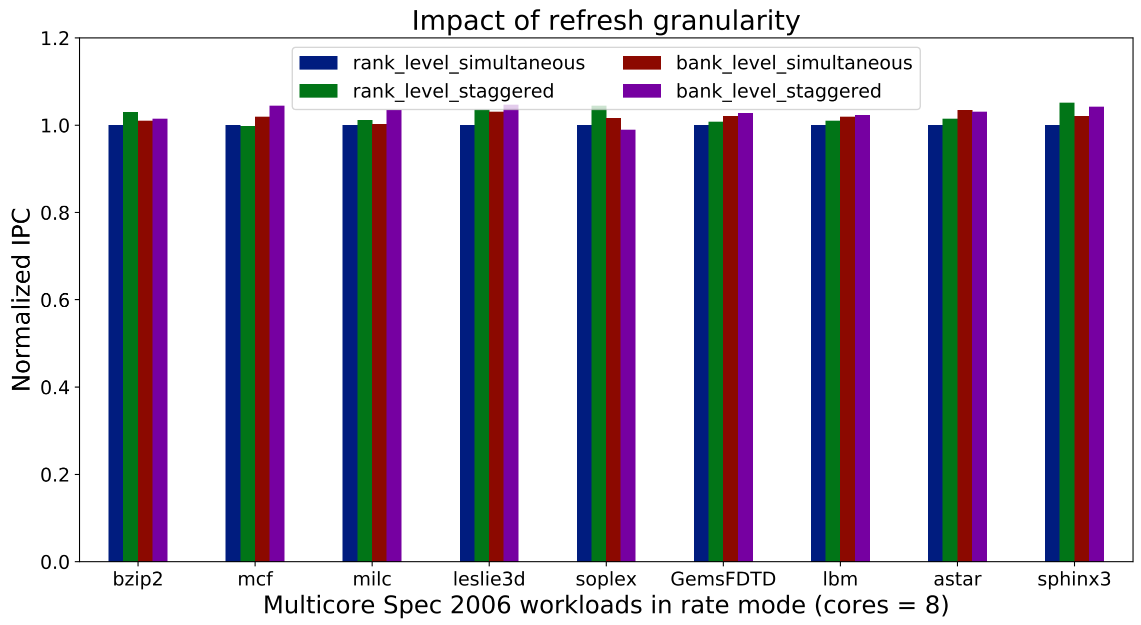

Figure 4.5: Impact of Refresh strategy

4.4 Dual command issue in HBM

The latest of all DRAM architectures studied, HBM, has a new feature where two commands can be issued over the command bus in the same clock cycle. The two commands cannot be arbitrary but need to be complimentary; i.e., if one is a row access command, the other should be a column access command. For workloads and DRAM architectures that are bottle-necked by the command throughput of the command bus, the ability to issue multiple commands per cycle could help increase performance.

Simulation studies are performed to quantify the performance improvement obtained by the dual-command feature of HBM when as compared to one without the dual command feature. Figure 4.6 shows that the dual command issue feature

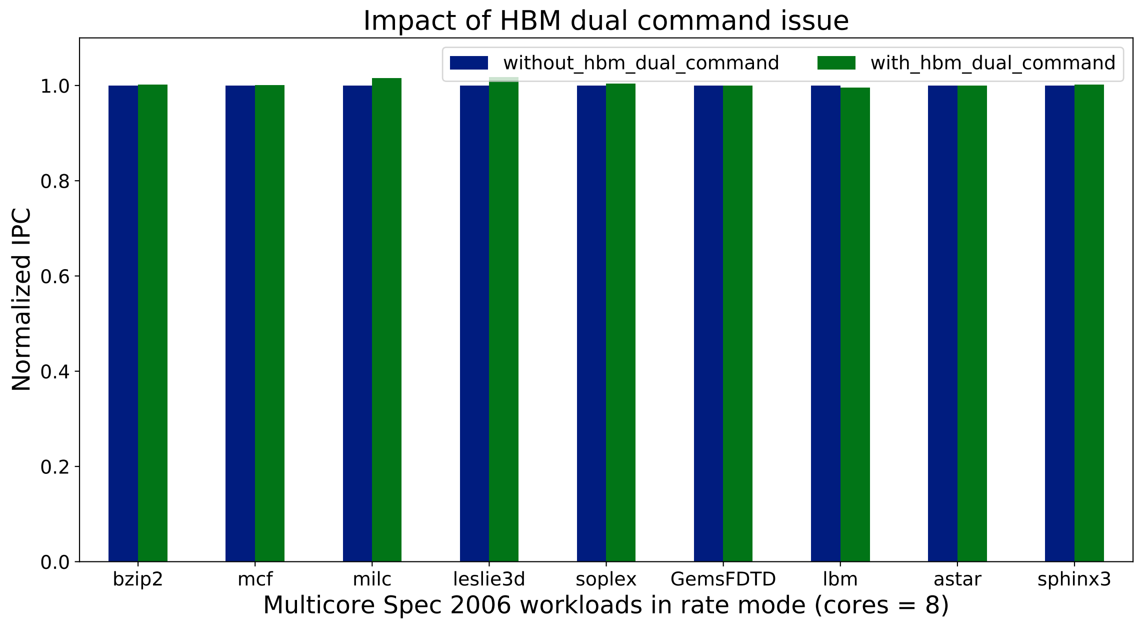

Figure 4.6: Impact of HBM dual command issue feature

in HBM give a marginal performance improvement of 0.4%, thereby implying that our workloads were not bound by the command bandwidth.

Chapter 5: Comparison of DRAM Architectures

This chapter analyzes the impact of the internal organization of DRAM architecture and the peripheral pin bandwidth on workload performance. DRAM DIMMs are primarily marketed in terms of their pin bandwidth. However, as we show in this chapter, the internal parallelism in these architectures plays a crucial role in achieving a sustained bandwidth close to the rated pin bandwidth. First, abstract studies are performed to determine the potential of different kinds of parallelism - such as channel level parallelism, rank level parallelism and bank level parallelism - in improving workload performance as well as the sustained memory bandwidth delivered by the memory subsystem. Next, some DRAM architectures belonging to different DDR protocols are compared for their relative performance and power trade-off. The abstract studies are performed for a DDR4-like base memory architecture by varying the number of channels, ranks and banks respectively.

5.1 Parallelism in DRAM Architectures - Abstract Study

Multiple degrees of parallelism exist in DRAM architectures. This section showcases how the degree of parallelism affects the ability to attain the peak supported pin bandwidth. The amount of parallelism that is actually possible in real designs is often constrained by the reality of physical design and the need to meet various timing constraints. Several circuit design constrains manifest in the form of the various DRAM protocol timing requirements. These timing constraints limit the performance improvement that the various parallelism can bring about. The protocol timing constraints together with the type and the amount of internal parallelism determine the latency and bandwidth characteristics of any DRAM architecture.

5.1.1 Channel level parallelism

While channel level parallelism is the most true form of parallelism available, the number of channels for off-package DRAM architectures is constrained by the number of output pins. However, on-package DRAM architectures such as HBM can have a much larger number of channels. In this subsection, the effect of the number of channels on workload performance is quantified for a set of memory intensive workloads.

The study is performed on a DDR4-like DRAM architecture with 2 ranks per channel and 8 banks per rank grouped into two bankgroups. Increasing the number of channels increases both the degree of memory parallelism available as well as the maximum available pin bandwidth. Figure 5.1 shows the normalized IPC performance improvement of having a higher number of channels. We observe the having number of channels equal to 2, 4, 8 increases the normalized IPC on average by about 19%, 29% and 33% respectively over a single channel system. For a four channel system, the improvement in normalized IPC ranges from 12%-58% for

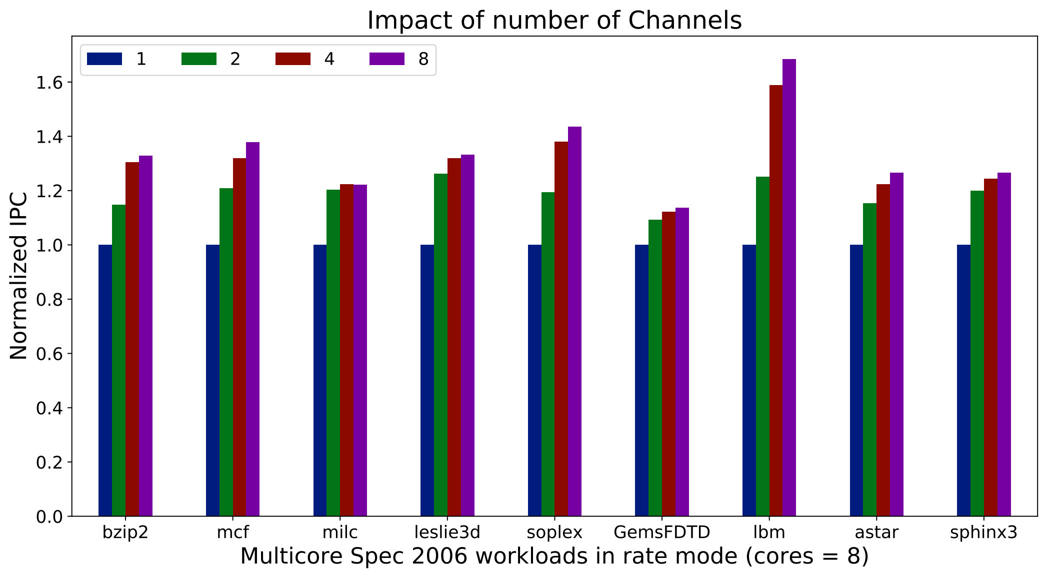

Figure 5.1: Normalized IPC Vs. Number of Channels

the set of workloads studied. An increase in the number of channels also increases the average bandwidth that the application can draw from the memory subsystem. Figure 5.2 shows the average bandwidth that is delivered by the memory system for different numbers of channels. The average memory bandwidth increases from 10 GB/s for a single channel system to 12 GB/s, 13 GB/s and 14 GB/s respectively for a memory system with number of channels equal to 2, 4, and 8. Also, Figure 5.3 shows the overall row buffer hit rates attained for different channel configurations. An increase in the number of channels increases the total number of available banks and thereby the number of rows that can all be open at the same. The row buffer hit rate percentage increases on average from only 34% for a single channel system to 48%, 62% and 73% when the number of channels are increased to 2, 4, and 8 respectively.

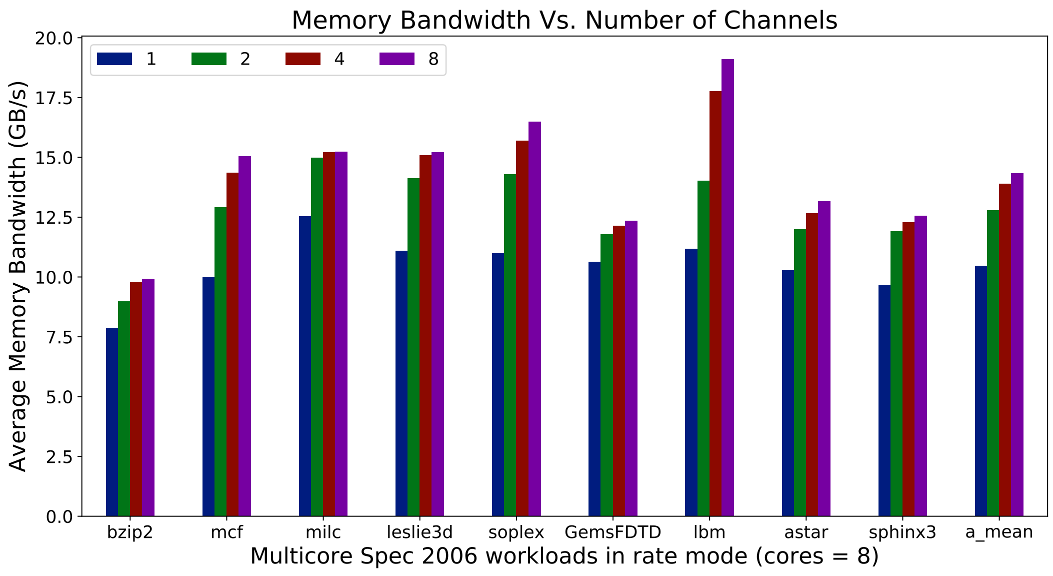

Figure 5.2: Average Memory Bandwidth Vs. Number of Channels

5.1.2 Rank level parallelism

A DRAM DIMM can consist of multiple ranks. Since all the ranks of a DIMM share the same peripheral pin interface, unlike channel-level parallelism, the data transfer from only one rank can happen at a time. However, seldom do any DRAM architectures achieve the rated pin bandwidth; i.e., the data transfer bandwidth across the peripheral interface is almost never the bottleneck. On the other hand, an increase in the number of ranks increases the total number of available banks and thereby increases the number of row buffers that could be open at the same time. Indirectly, this helps reduce the number of row conflicts and allows a greater number of memory accesses to be serviced with the latency closer to the minimum data transfer latency (tBURST). Overall, the number of memory accesses that could be readily pipelined to be drained across the data bus increases with increase in

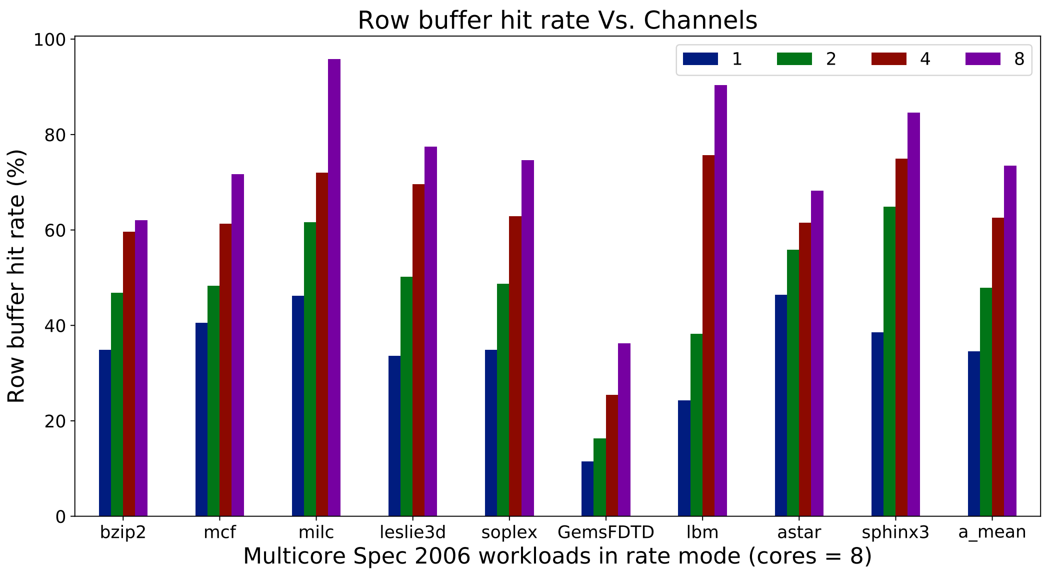

Figure 5.3: Row buffer hit rate Vs. Number of Channels

number of ranks.

One of the timing parameters that determines how effectively the Rank level parallelism can be utilized is the Rank-to-Rank turn around delay (tRTR). The tRTR is the additional time interval required between servicing read/write accesses belonging to different ranks. An increase in tRTR adversely affects workload performance as the access latency to service memory requests interleaved across different ranks increases.

Simulation studies are performed to quantify how an increase in rank-level parallelism impacts the workload performance. For a DDR4-like base system with 2 channels and 8 banks per rank grouped into two bankgroups, the number of ranks are increased from 1 to 2, 4 and 8. Figure 5.4 shows the normalized performance improvement of having a higher number of ranks. We observe that having 2, 4,

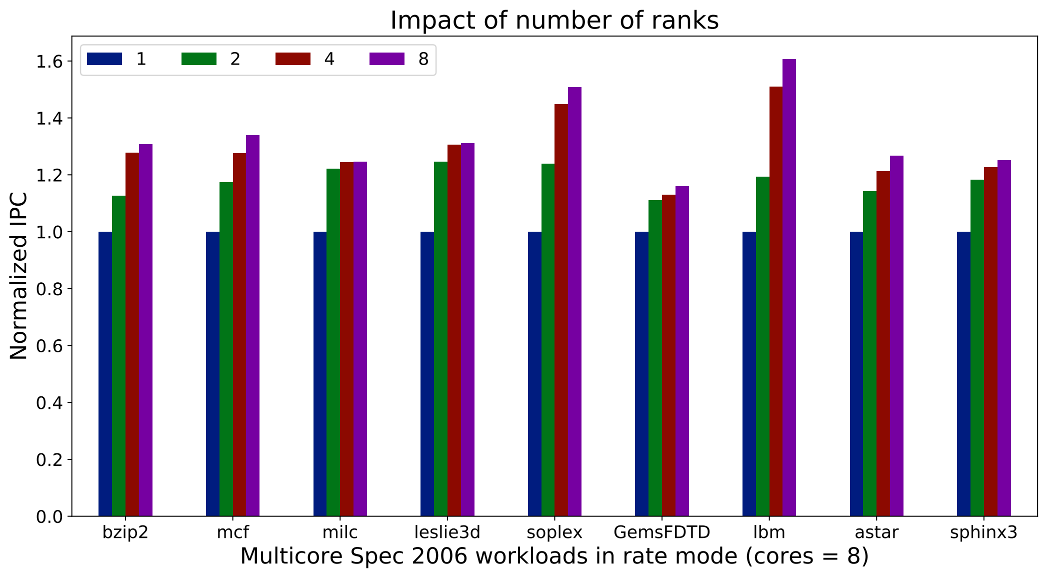

Figure 5.4: Normalized IPC Vs. Number of Ranks

8 ranks causes a normalized IPC improvement of 18%, 29% and 33% respectively w.r.t to the baseline single-rank system. The normalized IPC improvement of a two rank system ranges from 12% - 25% for the set of workloads studied. Figure 5.5 shows that the total average bandwidth delivered by the memory subsystem for the different rank configurations. Memory systems with number of ranks equal to 2, 4, 8 draw a average memory bandwidth of 13 GB/s, 14 GB/s and 15 GB/s respectively which is 20%, 31% and 35% greater than that of a single rank system. Figure 5.6 shows how much the overall row buffer hit rate increase with increase in number of ranks. A two rank system on average realizes 38% more row buffer hits than a single rank system.

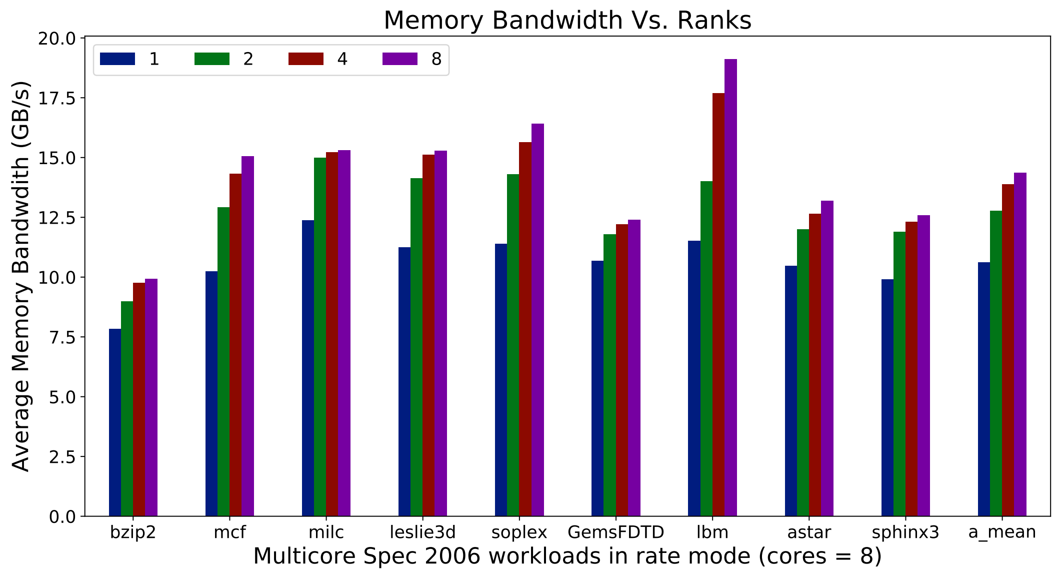

Figure 5.5: Average Memory Bandwidth Vs. Number of Ranks

5.1.3 Bank level parallelism

Each rank contains several banks. Similar to an increase in the number of ranks, an increase in the number of banks increases the number of row buffers that could all be open at the same time; and this reduces the possibility of row conflicts. A critical timing parameter that determines how effectively bank level parallelism can be utilized is the column-to-column access delay t_{CCD}. An increase in the value of t_{CCD} increases the time interval required between the servicing of two accesses belonging to the banks of a rank. Several other DRAM protocol timing constraints such as t_{RRD} (Row-to-Row activation delay) and t_{FAW} (Four window activation window constraint) limit the effective utilization of a bank level parallelism.

Simulation studies are performed to quantify how bank level parallelism affects workload performance. Figure 5.7 shows the normalized performance improvement

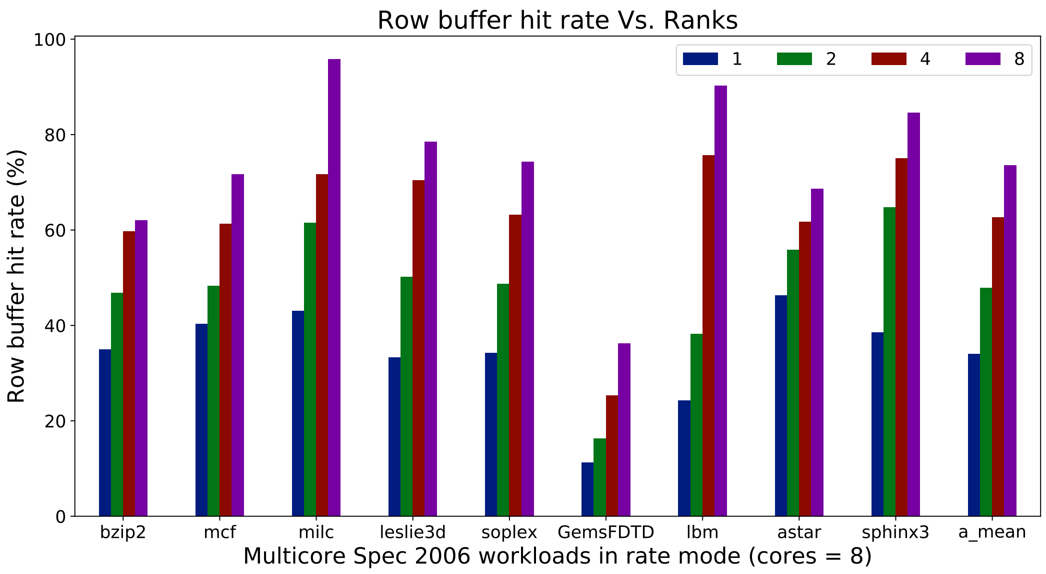

Figure 5.6: Row buffer hit rate Vs. Number of Ranks

of having a higher number of banks. Here, we normalize w.r.t the the baseline case of number of banks equal to 4. The figure shows that reducing the number of banks to 2, 1, causes a normalized IPC performance drop of 24% and 54% respectively. On the other hand increase the number of banks to 8, 16 and 32 increases the normalized IPC by 14%, 22% and 25% respectively.

Some of this behaviour could be explained by primarily looking at the row-buffer hit rate. Having a higher degree of parallelism allows for greater number of rows in different banks to be open at the same time, thereby helping to increase the row buffer hit rate. Fig 5.8 shows how the row buffer rate changes with the number of banks. Having a single bank causes the row buffer hit rate to drop to as low as 19%, whereas 4, 8 and 16 banks attain row buffer hit rates of 34%, 47% and 63% respectively. An increase in the row buffer hit rate and an increase in the degree

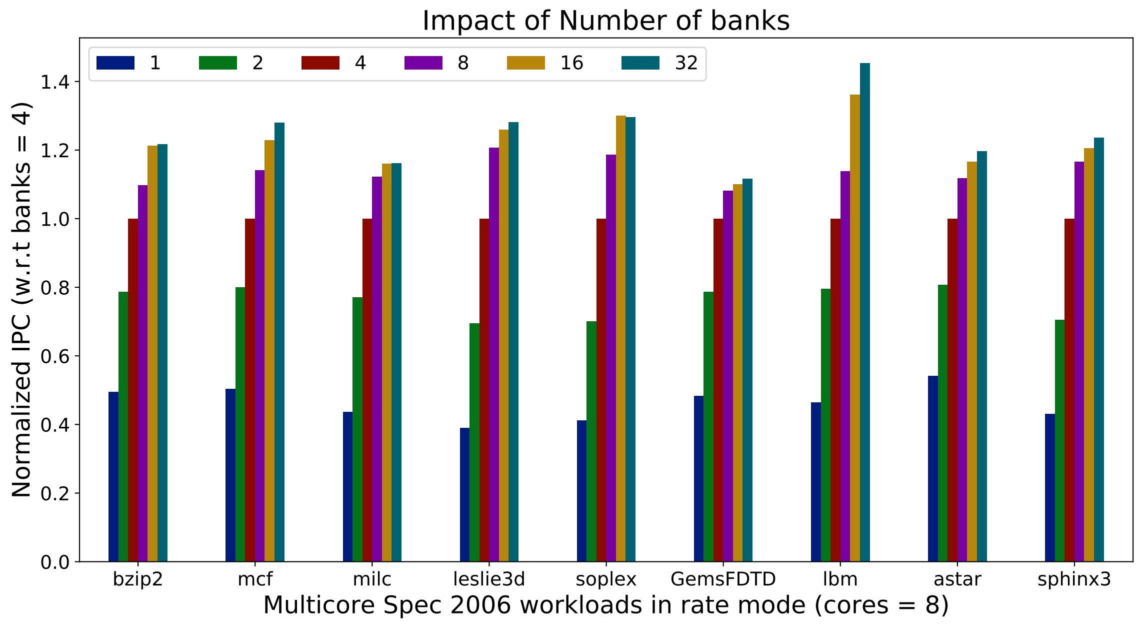

Figure 5.7: Normalized IPC Vs. Number of Banks

of parallelism helps increase the average bandwidth delivered. Figure 5.9 showcases the average bandwidth delivered by memory system configurations with different numbers of banks. Memory systems with 8 and 16 banks on average provide 14% and 16% more bandwidth than a 4-bank system.

5.2 Comparing DDR protocols - A focused study

In this section several DRAM architectures belonging to different DDR protocols are compared for their performance and power trade-off.

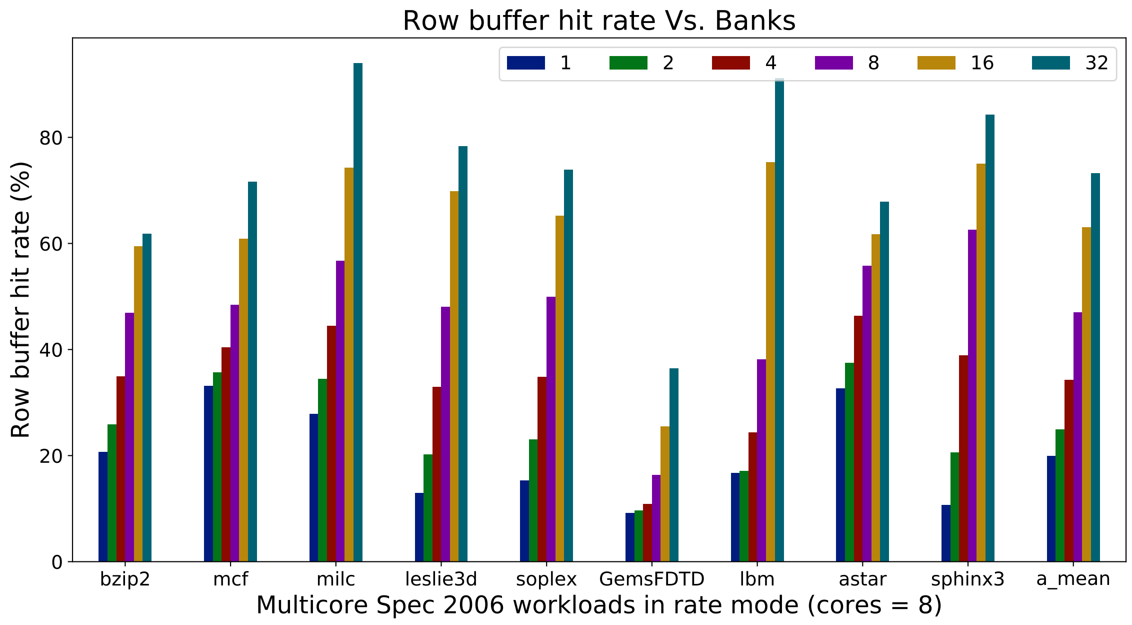

Figure 5.8: Row buffer hit rate Vs. Number of Banks

5.2.1 Performance comparison of DRAM architectures

The amount and type of parallelism available in DRAM architectures along with the protocol-specific timing constraints and rated pin bandwidth, together determine the impact of the memory subsystem on application performance. While directly comparing DRAM architectures belonging to different DDR protocols is not an apples-to-apples kind of comparison, it does however showcase how the different DRAM architectures affect the overall application performance. Table 5.1 lists the various DRAM architectures studied. It lists the DDR protocol the DRAM architectures belongs to, the degree and type of parallelism available, as well as the frequency of the data bus. The internal parallelism is specified in terms of the number of channels, number of ranks per channel, number of bankgroups into which

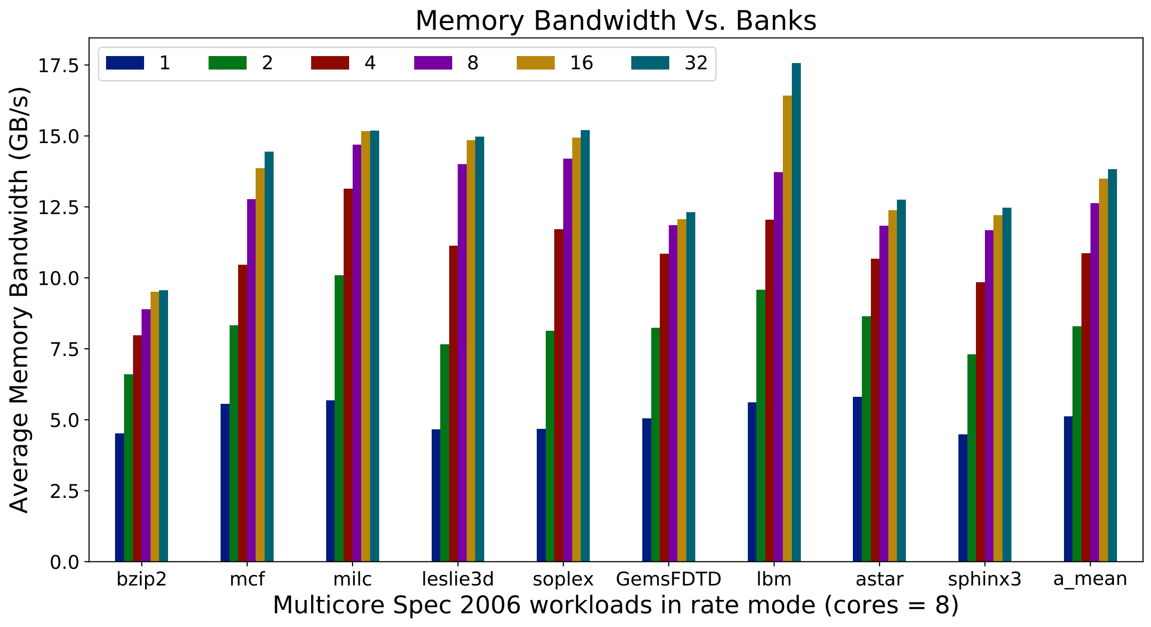

Figure 5.9: Average Memory Bandwidth Vs. Number of Banks

the banks of the rank are divided and the total number of banks per rank. Figure 5.11 shows the average bandwidth delivered by each of these DRAM architectures. We observe that, for the studied workloads, the internal parallelism is the primary contributing factor in determining the average bandwidth. The contribution of the frequency of the data transfer bus in increasing the sustained bandwidth delivered is minimal for the set of workloads studied. For example, DDR4_3200 architecture only has a marginal 1% more average bandwidth than DDR4_1600 architecture, while HBM_2000 delivers a meagre 4% more average bandwidth than HBM_1000.

Figure 5.10 shows the normalized IPC for the workloads across different DRAM architectures. The two HBM DRAM architectures produces a normalized IPC improvement of 21% and 26% respectively over the DDR_1600 baseline. Overall, we observe that for two DRAM architectures belonging to the same protocol and with

Table 5.1: Configuration of DRAM architectures studied showing the type of DDR protocol, data bus frequency and the degree of internal parallelism

Name	DDR Protocol	Frequency (in MHz)	Channels	Ranks	Bankgroups	Banks per Rank
LPDDR3_1333	LPDDR3	1333	2	1	1	8
LPDDR3_1866	LPDD3	1866	2	1	1	8
DDR3_1600	DDR3	1600	2	1	1	8
DDR3_1866	DDR3	1866	2	1	1	8
DDR4_1866	DDR4	1866	2	1	4	16
DDR4_3200	DDR4	3200	2	1	4	16
HBM_1000	HBM	10000	16	1	4	16
HBM_2000	HBM	2000	16	1	4	16

the same degree of internal parallelism, the performance improvement with an increase in frequency of the data bus is only marginal. DDR4_3000 architecture shows only a marginal normalized IPC improvement of 2% over DDR4_1866.

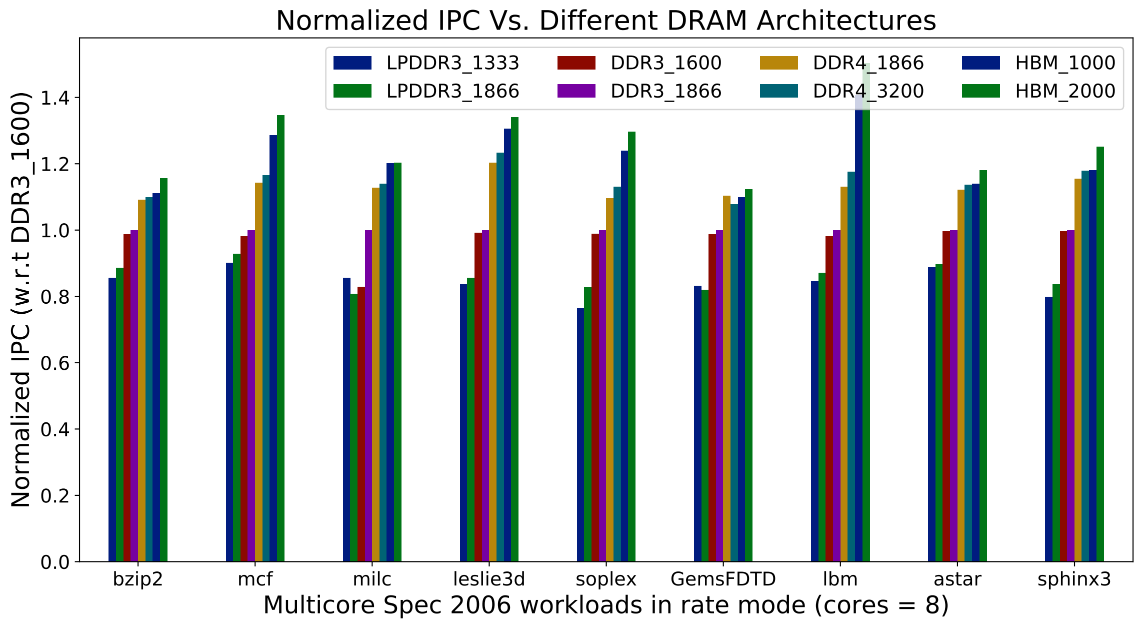

Figure 5.10: Normalized IPC for DRAM architectures belonging to different DDR protocols

5.2.2 Energy and Power comparison of DRAM architectures

Figure 5.12 shows the dynamic energy consumption for the workloads across the different DRAM architectures. We observe that that the dynamic energy consumption increases significantly with an increase in the frequency of the data bus. DDR4_3200 architecture has much higher energy consumption than DDR4_1866. Pareto plots showing the variation of power and CPI for the different DRAM architectures are drawn for all the benchmarks in Figure 5.13. We observe that the HBM and LPDDR architectures are Pareto optimal in terms of power and performance. For architectures belonging to the same DDR protocol with the same degree of internal parallelism, a higher data-bus bandwidth gives only a marginal improve-

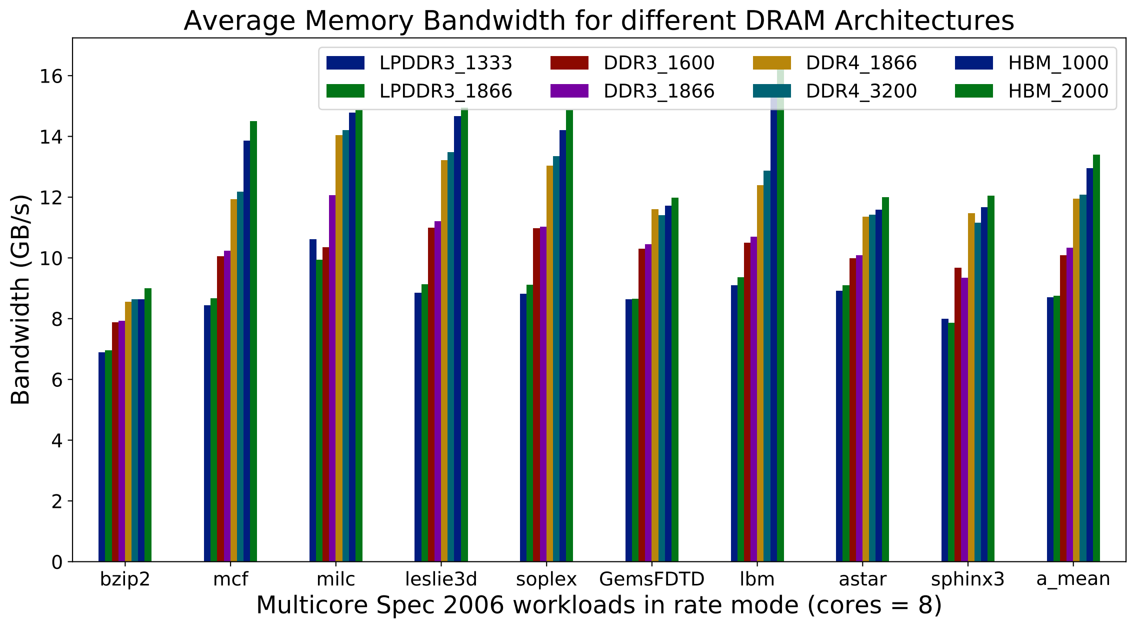

Figure 5.11: Average Bandwidth for DRAM architectures belonging to different DDR protocols

ment in CPI while consuming significantly more power. For example, HBM_2000 architecture on average has 4% lesser CPI than HBM_1000 but consumes 70% more dynamic power. We conclude that for the set of memory intensive workloads studied, the internal parallelism of the DRAM architectures plays a far more important role in improving workload performance than the bandwidth of the data-bus. In addition, the power consumption associated with higher data-bus bandwidths is not commensurate with the improvement in workload performance.

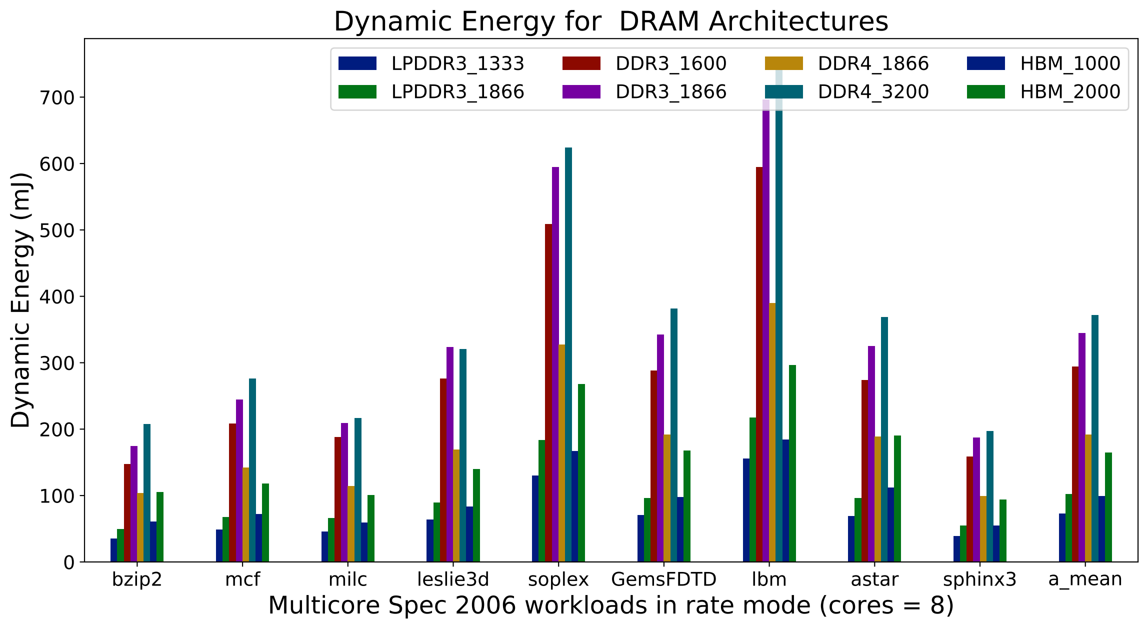

Figure 5.12: Dynamic energy for DRAM architectures belonging to different DDR protocols

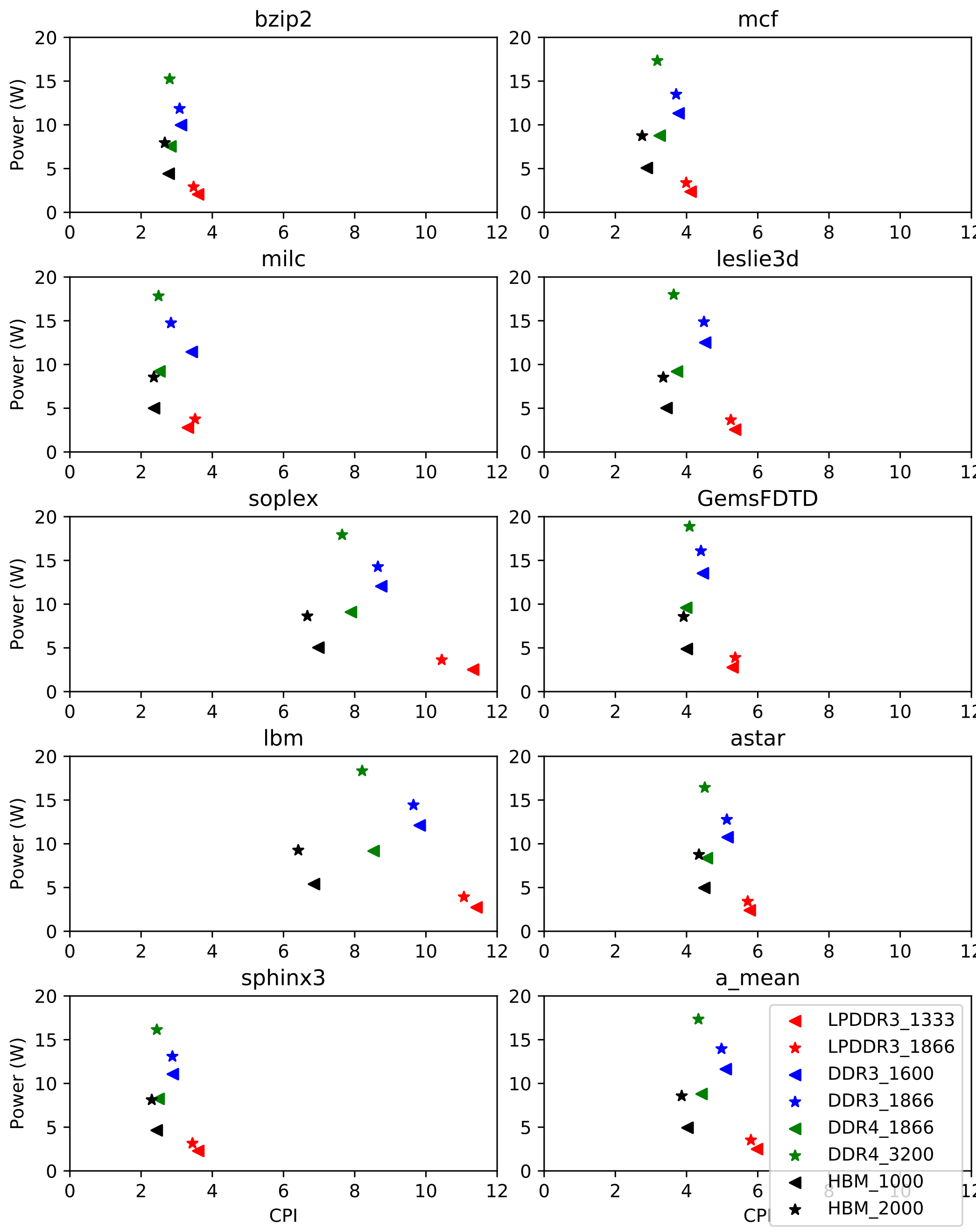

Figure 5.13: Pareto plots showing power and CPI for different DRAM architectures

Chapter 6: Conclusion and Future Work

This thesis describes the design and building of a DRAM memory simulator. It also studies different DRAM architectures comparing their parallelism and bandwidth trade-offs. It showcases the merits of some of the newer DRAM technologies both in terms of power and performance. It also demonstrates the importance of internal DRAM parallelism and showcases how it is essential to attain an average bandwidth that is closer to the maximum supported pin bandwidth. It also presents the experiences in designing a DRAM simulator and integrating it with different front-end CPU simulators.

6.1 Future Work

Since the DRAM pin bandwidth is not a very good indicator of the performance that a memory system will deliver, we believe that it would be worthwhile to investigate a new metric which would take into consideration the various types and degree of internal parallelism along with the pin bandwidth to give a performance number that better correlates with the IPC seen for the workloads. In addition, since the pin bandwidth is under utilized in most DRAM architectures it would be worth investigating the potential of larger cache block sizes for the last-level-cache or deeper prefetching to draw greater useful bandwidth from the memory sub-system. Instead of each memory access fetching only 64B of data to fill a cache block in the last level cache, if it were to fetch a larger amount of data per memory access, the bandwidth of the memory data bus could be better utilized. A study that looks at the various DRAM architecture parameters, memory controller parameters and design of cache hierarchy together could be useful in determining the optimal prefetch depth.

Bibliography

[1] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu. Knights landing: Second-generation intel xeon phi product. *IEEE Micro*, 36(2):34–46, Mar 2016.

[2] JEDEC. *DDR3 SDRAM Standard, JESD79-3*. JEDEC Solid State Technology Association, June 2007.

[3] JEDEC. *DDR4 SDRAM Standard, JESD79-4*. JEDEC Solid State Technology Association, September 2012.

[4] JEDEC. *High Bandwidth Memory (HBM) DRAM, JESD235A*. JEDEC Solid State Technology Association, November 2015.

[5] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh. Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 126–136, Feb 2015.

[6] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim. Transparent hardware management of stacked dram as part of memory. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 13–24, Dec 2014.

[7] JEDEC. *Low Power Double Data Rate 3 (LPDDR3), JESD209-3*. JEDEC Solid State Technology Association, May 20012.

[8] JEDEC. *Low Power Double Data Rate (LPDDR4), JESD209-4A*. JEDEC Solid State Technology Association, November 2015.

[9] JEDEC. *Graphics Double Data Rate (GDDR5) SGRAM Standard, JESD212C*. JEDEC Solid State Technology Association, February 2016.

[10] Bruce Jacob, Spencer Ng, and David Wang. *Memory Systems: Cache, DRAM, and Disk*. Morgan Kaufmann, 2007.

[11] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[12] Daniel Sanchez and Christos Kozyrakis. Zsim: Fast and accurate microarchitectural simulation of thousand-core systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 475–486, New York, NY, USA, 2013. ACM.

[13] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 52:1–52:12, New York, NY, USA, 2011. ACM.

[14] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. The structural simulation toolkit. *SIGMETRICS Perform. Eval. Rev.*, 38(4):37–42, March 2011.

[15] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. Dramsim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters*, 10(1):16–19, Jan 2011.

[16] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Computer Architecture Letters*, 15(1):45–49, Jan 2016.

[17] I. Bhati, Z. Chishti, S. L. Lu, and B. Jacob. Flexible auto-refresh: Enabling scalable and energy-efficient dram refresh reductions. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 235–246, June 2015.

[18] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. A performance comparison of contemporary dram architectures. In *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*, pages 222–233, 1999.

[19] K. K. W. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu. Improving dram performance by parallelizing refreshes with accesses. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 356–367, Feb 2014.