ABSTRACT

Title Of Dissertation: UNDERSTANDING AND OPTIMIZING
HIGH-SPEED SERIAL MEMORY SYSTEM ARCHITECTURES

Brinda Ganesh, Doctor of Philosophy, 2007

Dissertation Directed by: Professor Bruce Jacob
Department of Electrical and Computer Engineering

Performance improvements in memory systems have traditionally been obtained by scaling data bus width and speed. Maintaining this trend while continuing to satisfy memory capacity demands of server systems is challenging due to the electrical constraints posed by high-speed parallel buses. To satisfy the dual needs of memory bandwidth and memory system capacity, new memory system protocols have been proposed by the leaders in the memory system industry. These protocols replace the conventional memory bus interface between the memory controller and the memory modules with narrow, high-speed, uni-directional point-to point interfaces. The memory controller communicates with the memory modules using a packet-based protocol, which is translated to the conventional DRAM commands at the memory modules.

Memory latency has been widely accepted as one of the key performance bottlenecks in computer architecture. Hence, any changes to memory sub-system architecture and protocol can have a significant impact on overall system performance. In the first part of this dissertation, we did an extensive study and analysis of how the behavior of newly proposed memory architecture to identify clearly how it impacts memory sub-system performance and what the key performance limiters are. We then went on to use the insights we gained from this analysis to propose two optimization techniques focussed on improving the performance of the memory system.

We first evaluated the performance of the current *de facto* serial memory system standard, FBDIMM (Fully Buffered DIMM) with respect to the conventional wide-bus architectures that have been in use for decades. We found that the relative performance of a FBDIMM system with respect to a conventional DDRx system was a strong function of the bandwidth utilization, with FBDIMM systems doing worse in low utilization systems and often out-performing DDRx systems at higher system utilizations. More interestingly, we found that many of the memory controller policies that have been in use in DDRx systems performed similarly on a FBDIMM system.

Memory latency typically has a significant impact on overall system performance. FBDIMM systems, by using daisy chaining and serialization, increase the default latency cost of a memory transaction. In a longer memory channel, i.e. a channel with 8 DIMMs of memory, inefficient link utilization and memory controller scheduling policies can contribute to a further reduction in system performance. We propose two main optimization techniques to tackle these inefficiencies - reordering data on the return link and buffering at the memory module. Both these policies lower read latency by 10-20% and improve application performance by 2-25%.

UNDERSTANDING AND OPTIMIZING
HIGH-SPEED SERIAL MEMORY SYSTEM PROTOCOLS

by

Brinda Ganesh

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

Doctor of Philosophy

2007

Advisory Committee:

Professor Bruce Jacob, Chair
Professor Shuvra Bhattacharyya
Dr. Xiaowei Shen
Professor Ankur Srivastava
Professor Lawrence Washington
Professor Donald Yeung

To

Ankush, Mummy and Daddy

You are simply the best!!

ACKNOWLEDGEMENTS

This thesis would not have been possible if it were not for a number of people with whom I have shared this journey through graduate school. First of all, I would like to thank my wonderful husband, Ankush who has been on my side the entire way. He has made me laugh, proof read my papers so many times that he knows the details of my own research more than me, accompanied on numerous hikes and bike rides and most importantly believed in me throughout.

Most importantly, I would like to thank my wonderful family who have been patient, supportive and very understanding through my years in graduate school. My parents have been one of my biggest cheerleaders, while I went through the ups and downs of graduate school. I would also like to thank my aunt, Kalpanachikkamma and my grandmother, Ammamma who have given me a home to go to in the United States. I have, thanks to both their efforts, not ever felt alone in this country or far from my family.

My advisor, Dr. Bruce Jacob, has provided me invaluable guidance and advice through all my years in graduate school. He has taught me much of what I know about how to do research, how to write papers and how to give presentations. I would really like to thank him for his patience and support, while I took my time finding my way out of graduate school.

My mentor at IBM, Xiaowei Shen, has been one of the best mentors anybody could ask for. Xiaowei is an insightful researcher, a great mentor and wonderful person to work with. Xiaowei has been largely instrumental in me getting sufficient momentum to get done with the final disseratation. While at IBM, I also had the pleasure of working with Jessica Tseng, who was always ready to discuss everything from issue queue architectures to what to do in New York City. Thanks Jessica for my fun time at Watson and for opening my mind about how to approach problems!! Thanks also to Dustin Vanstee, with whom I had very interesting discussions on memory protocol choices.

Thanks to my committee members, Dr. Donald Yeung, Dr. Shuvra Bhattacharyya, Dr. Ankur Srivastava and Dr. Lawrence Washington, for their feedback and many suggestions which have been invaluable.

Many thanks to Wanli who helped me with porting DRAMsim to M5 and answered all my ridiculous questions cheerfully. Thanks also to Men-ju Wu for his help with getting checkpoints to work and Jayanth Gummaraju for providing me his DRAMsim port to the M5 simulator.

While at Maryland, I got to work with many wonderful folk who made my days in the SCA Lab a fun experience. Aamer Jaleel has always been available to discuss computer architecture, life and to roller blade down Paint branch trail. Thanks Aamer for all the fun times in the good old SCA lab. I am also very grateful to David Wang, who has taught me much of what I know about memory system architectures and has been always willing to explain the same thing to me multiple times. I shall definitely miss Ohm, who was always around to take off to watch cartoon movies with, discuss Korean series and the bug currently irking me. Thanks to Sada for his willingness to discuss any idea of mine no matter how half-baked. I also had a lot of fun in the lab thanks to many folks, in no particular order, Anasua, Iyerb, Sumesh, Surupa, Rania, Xuanhua, Choi, Lei, Mainak,Mike and Hima.

Ayan and Ayush, with whom I have shared the Ph.D. journey have always been willing to provide me with a cup of coffee, listen to my troubles and help me out. Thanks to Ayush for constantly challenging my boundaries with arguments and discussions about fair-trade coffee, free speech and workers rights and to Ayan who tolerated these heated discussions and on other occasions added fuel to the fire. Avanti has been a wonderful room-mate and friend. I will always cherish the times we spent laughing our daily troubles off, staying up well past our bed-time watching Jay Leno and for having, at least temporarily, helped me maintain a squeaky clean house. Thanks also to Andre, Gelareh, Anubha, Anu, Soumya, Jo for fun-times at Maryland.

Table Of Contents

List of Figures

List of Tables

Chapter 1: Introduction

The growing size of application working sets, the popularity of software-based multimedia and graphics workloads, and the increased use of speculative techniques have contributed to the rise in bandwidth and capacity requirements of computer memory sub-systems. Capacity needs have been met by building increasingly dense DRAM chips and bandwidth needs have been met by scaling front-side bus data rates and using wide, parallel buses. However, traditional bus topologies have reached a point where they fail to scale well into the future.

The electrical constraints of high-speed parallel buses complicate bus scaling in terms of loads, speeds, and widths [1]. Consequently the maximum number of DIMMs per channel in successive DRAM generations has been *decreasing*. SDRAM channels have supported up to 8 DIMMs of memory, some types of DDR channels support only 4 DIMMs, DDR2 channels have but 2 DIMMs, and DDR3 channels are expected to support only a single DIMM. In addition, the serpentine routing required for electrical path-length matching of the data wires becomes challenging as bus widths increase. For the bus widths of today, the motherboard area occupied by a single channel is significant, complicating the task of adding capacity by increasing the number of channels. To address these scalability issues, an alternate DRAM technology, the Fully Buffered Dual Inline Memory Module (FBDIMM) [3] has been introduced.

The FBDIMM memory architecture replaces the shared parallel interface between the memory controller and DRAM chips with a point-to-point serial interface between the memory controller and an intermediate buffer, the Advanced Memory Buffer (AMB). The on-DIMM interface between the AMB and the DRAM modules is identical to that seen in

DDR2 or DDR3 systems, which we shall refer to as DDRx systems. The serial interface is split into two uni-directional buses, one for read traffic (northbound channel) and another for write and command traffic (southbound channel), as shown in Fig 1.1. FBDIMMs adopts a packet-based protocol that bundles commands and data into frames that are transmitted on the channel and then converted to the DDRx protocol by the AMB.

Understanding FBDIMM. The FBDIMMs' unique interface raises questions on whether existing memory controller policies will continue to be relevant to future memory architectures. In particular in the first part of the dissertation we ask the following questions:

- How do DDRx and FBDIMM systems compare with respect to latency and bandwidth?
- What is the most important factor that impacts the performance of FBDIMM systems?
- How do FBDIMM systems respond to the values of parameters like row buffer management policy, scheduling policy and topology?

To answer these questions, we did an evaluation of the performance of FBDIMM systems and DDRx systems using memory inputs from the SPEC 2000[2] workload suite.

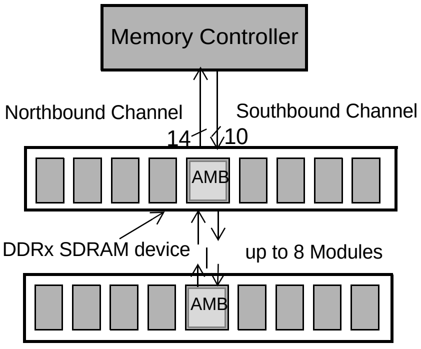


Figure 1.1. FBDIMM Memory System. In the FBDIMM organization, there are no multi-drop busses; DIMM-to-DIMM connections are point-to-point. The memory controller is connected to the nearest AMB, via two uni-directional links. The AMB is in turn connected to its southern neighbor via the same two links.

Our study showed that the relative performance of a FBDIMM system and a DDRx system was a strong function of the bandwidth utilization of the input streams. Overall, the FBDIMM system had a 27% average higher latency, which were mainly due to workloads with bandwidth utilizations of less than 50% total DDRx DRAM bandwidth. This latency degradation becomes a latency improvement of nearly 10% for FBD-DDR3 systems as the application bandwidth utilization increased past 75% due to the ability of the FBD memory controller to use the additional DRAM level parallelism, split bus architecture and ability to send multiple DRAM commands in the same clock cycle. However, the additional system bandwidth available in a FBDIMM system resulted in an average of approximately 10% improvement in overall bandwidth with most of the benefits coming again for workloads with higher bandwidth utilization.

More interestingly, we found that the scheduling policies and row buffer management policies used in DDRx systems continued to perform comparably in FBDIMM systems. In both cases, a scheduling policy that prioritizes read traffic over write traffic had the best latency characteristics for both open page and closed page systems. A scheduling policy that prioritized traffic to currently open banks in the system had the best bandwidth characteristics while a greedy approach did very well in closed page systems.

One difference that we found with regard to FBDIMM and DDRx system behavior was their response to the use of posted CAS, a DRAM protocol feature which simplifies memory controller design by allowing the memory controller to bundle a row activation and read/write command in back-to-back command cycles. Unlike DDRx systems, FBDIMM systems using posted CAS had worse latency and bandwidth characteristics than systems which did not use this. This difference arose from the organization and use of the

FBDIMM command frame and the sharing of the FBDIMM southbound bus by commands and write data.

Detailed measurements of the contributors to the read latency of a transaction revealed that a significant factor contributing to the overall latency was delays associated with the unavailability of memory system resources, such as southbound channel, northbound channel and DRAM. Scaling the memory system configuration, by adding more ranks or channels, resulted in the unavailability of each of these factors varying in a different fashion and interacting in different ways to impact the observed latency. In general, we observed that short channel FBDIMM systems are limited by DRAM availability, while long channel FBDIMM systems are bound by channel bandwidth. This problem is exacerbated in variable latency mode configurations where significant latency and bandwidth degradation occur due to inefficient usage of the northbound FBDIMM channel.

Optimizing FBDIMM Read Latency. The use of serialization and a multi-hop topology in FBDIMM systems has led to increases in the default cost of a read transaction. The FBDIMM protocol allows the channel to be configured in one of two modes, a fixed latency mode where the round-trip latency for a transaction is identical for all DIMMs and is set to the round-trip latency of the last DIMM in the chain. This mode imposes a higher default latency cost which is not desirable. The alternate mode, known as the variable latency mode, has been provided to target this latency cost. In this mode the channel is configured such that the round-trip latency of a transaction is a function of the distance of the DIMM from the memory controller. By allowing this, the FBDIMM protocol hopes to lower overall read latency.

Although the variable latency mode is able to reduce the average read latency for many of the workloads studied, this was not always true for applications executing in longer FBDIMM channels where the latency reductions were most needed. This problem was due to two reasons, one the inefficient utilization of the north link and the second that transactions to closer DIMMs have to often wait for read data from further DIMMs to complete using the north link although the DRAM is ready.

We studied two techniques to improve the latency of an FBDIMM channel using the following techniques

- Out of order return of read data or allowing read data to return out of order,
- Buffers at the AMB for north link data to permit transactions to nearer DIMMs being issued in advance

Like previously defined memory protocols, the FBDIMM specification assumes an omniscient memory controller that manages all system resources, including the DRAM, on-DIMM buses i.e. command and data bus, and on-board links i.e. the south and north links. The memory controller has to guarantee that there are no violations of DRAM timing parameters or any conflicts on any of the various system buses. The default implementation of the protocol assumes that read data returns in the order that it was scheduled. Consequently, the scheduler is unable to take advantage of idle time on the north links which occur prior to the use of the link by a previously scheduled read transaction.

The first optimization that we looked at was to improve latency and bus utilization by relaxing the need for data to return in the same order as the commands. We propose a technique to permit re-ordering of returning read data that can be implemented without modifying the existing FBDIMM memory protocol. Permitting reordering of read data improves the maximum sustainable bandwidth of the system by 1-2 GBps. Application latency improves in a 4-8 DIMM deep system by an average of 5-25% while bandwidth utilization increased by 5-10%. Multi-program workload runs on a full system simulator demonstrate that this technique improves overall IPC by an average of 5-15%, with the most benefits being seen in a system with longer channels.

The second technique that we explored was focussed on de coupling DIMM availability from north link availability. We proposed using buffers on the northbound channel pat that hold read data frames. Buffering enables the memory controller to read data out of the DRAM rows without having to wait for the north link to become available. We examined several buffering policies including Source Buffering, Global Variable Buffering and Global Binary Buffering. These policies are distinguished by where buffering is permitted, i.e. at the DIMM supplying the data or at any DIMM on the path from the DIMM to the memory controller, and the buffering duration specified, a pre-defined fixed duration or a dynamically determined value. Buffering is managed by the memory controller and all buffering durations are specified by the memory controller in the command frames.

We found that buffering results in a speed-up of 2 to 25% for memory system topologies with 4 and 8 DIMMs per channel. Again, most of these benefits arose from increasing the ability of the memory controller to move data out of the closest DIMM earlier. A memory controller that uses buffering and re-ordering of data returns further reduces memory latency by an additional 10%. The overall speed-up by using both optimizations is on the order of 2-25%, over the baseline system which only allows in order return of data and does not support buffering.

This dissertation is organized as follows. Chapter 2 describes the past work done in the memory system area both in academia and industry. Chapter 3 describes the evolution of memory architectures and describes in detail the FBDIMM memory architecture and protocol. Chapter 4 has a comparison of the characteristics of DDRx and FBDIMM systems. It also has detailed results and analysis of the behavior of FBDIMM memory systems as system configurations are scaled. Chapter 6 describes the optimizations that we explored to improve the average read latency of a FBDIMM memory system. Chapter 7 summarizes the final conclusions of this dissertation.

Chapter 2: Related Work

Microprocessor speeds have tracked Moore's law [1], doubling every eighteen months, while DRAM speeds have increased at a more moderate rate of roughly 7% [2], doubling only every 10 years. The resulting gap, also termed the memory wall, increases at 50% every year. The chief consequence of this has been the development of techniques to reduce or hide memory latency.

At the architectural level these techniques including lock-up free caches [3], hardware and software pre-fetching [4, 5], speculative execution and multi-threading focus on tolerating memory latency. Burger et al. [6] demonstrated that the majority of these techniques lowered latency by increasing bandwidth demands. A later study which examined different DRAM architectures by Cuppu et al. [7] demonstrated that memory manufacturers were able to meet bandwidth demands but had not been able to tackle latency effectively.

2.1. Performance Optimizations for the Memory Sub-System

There have been several studies at the controller level which examine how to lower latency while simultaneously increasing bandwidth utilization. The focusses of these techniques have been lowering row-buffer miss rates by employing address mapping, memory request access reordering or split-transaction scheduling. Row-buffer misses are expensive, because conflicts can be resolved only after a precharge-activate sequence. Zhang et al [8] studied how address mapping can be used to lower row-buffer conflicts are reduced. The scheme attempts to distribute blocks that occupy the same cache set across multiple banks in the system, by xoring the lower page-id bits with the bank-index bits. Rixner et al [9] studied how re-ordering accesses at the controller to increase row buffer hits can be used to lower latency and improve bandwidth for media processing streams. They studied several policies that re-ordered requests based on age, arrival order, type i.e. loads over stores and ratio of column to row accesses. [10] studied how such re-ordering benefitted from the presence of SRAM caches on the DRAM aka Virtual-Channel DRAM for web servers Takizawa et al.[11] proposed a memory arbiter that increased the bandwidth utilization by reducing bank conflicts and bus turnarounds in a multi-core environment. The arbiter reduces bank conflicts by reducing the priority of DRAM accesses that are to the same bank as the previously issued access or if the access direction (read or write) is different from that of the previously issued access.

Natarajan et al [12] studied how memory controller policies for row buffer management policies, including open page, closed page and delayed closed page policies and command scheduling impacts latency and sustained bandwidth. They show that access re-ordering with a closed page policy provides the best bandwidth and latency for DDR/DDR2 based systems. They also demonstrate in DDR/DDR2 that intelligent read-write switching is influential in reducing bus inefficiencies.

Wang [13] proposed a memory request re-ordering algorithm which focussed on increasing bandwidth utilization. The algorithm attempted to get around bus constraints like bus turnaround time, and DRAM constraints like row-activation windows. Shao et al [14] propose a burst reordering scheduling scheme to improve the system memory bus utilization. The scheme reorders memory requests, such that read accesses, that are addressed to the same row of the same bank are clustered together. Writes are typically delayed until the write queue is either full or hits a particular threshold size. When the latter occurs, the scheduler piggybacks write transactions onto the ongoing burst by issuing a write transaction which is addressed to the currently open row. When the write queue is full, the scheduler issues the oldest write transaction in the system.

Lin et al [15] studied how memory controller based pre-fetching can lower the system latency in a system with an on-chip memory controller. Zhu et al [16], on the other hand studied how awareness of resource usage of threads in an SMT could be used to prioritize memory requests.

Cuppu et al [17] demonstrated that concurrency is important even in a uni-processor system, but split-transaction support would lower latency of individual operations. [18] studied how split-transaction scheduling in a multi-channel environment could be used to lower latency.

Intelligent static address mapping techniques [15] [8] have been used to lower memory latency by increasing row-buffer hits. The Impulse group at University of Utah [19] proposed adding an additional layer of address mapping in the memory controller. This mapping technique reduced memory latency by mapping non-adjacent data to the same cacheline and thus increasing cacheline sub-block usage. The mapping is handled by the memory controller with information from the operating system. They used this mapping in conjunction with a parallel vector access unit [20], which enabled the memory controller to encode multiple requests in a single command to improve bus utilization.

Shao et al. [21] proposed a bit-reversal address mapping scheme for SDRAM systems. The scheme reverses the 'v' highest address bits and uses these to map the rank bits, bank bits and part of the row address bits. They demonstrate that this scheme improves execution time by mapping the most likely changing bits to the column, rank and bank bits and by redistributing memory accesses to be equally distributed across all banks.

Mitra et al.[22] characterized the behavior of 3D graphics workloads to understand the architectural requirements for these applications. They explored the impact of using architectural optimizations such as active texture memory management, speculative rendering and dynamic tiling on the performance of graphics applications. In addition they characterized the memory bandwidth requirements for these applications.

Embedded system controllers used in media systems have to provide high bandwidth utilization for the media and signal processing workloads while simultaneously providing low latency service to on-chip processing elements. Harmsze et al [23] proposed a solution to this problem in which they allocate fixed scheduling intervals to continuous streams and any additional slack time at a higher priority to CPUs and peripherals. This scheme was used in conjunction with on-chip buffering to provide compile-time guarantees of performance. This scheme does not take into account the state of the underlying DRAM. Lee et al. [24, 25] proposed a memory controller design that used a layered architecture, with a layer dedicated to DRAM management, QoS scheduling and address generation for continuos streams requestors to solve the same problem. The DRAM management layer generated the DRAM command stream required to process an actual request. As in earlier work, the DRAM layer designed a schedule that takes into account bank conflicts, bus turnaround times etc. In addition, the Quality of Service Access layer provided the DRAM layer with information regarding the priority of a given request which is taken into account to build the schedule. The QoS Access layer sends the DRAM layer information whether a given access is latency-sensitive, bandwidth sensitive or neither. Like Harmsze et al, they provide fixed bandwidth to a bandwidth sensitive stream, but unlike them they build in pre-emptive mechanisms which allows the scheduler to pre-empt a bandwidth-sensitive stream when a latency sensitive requestor makes a request.

Nesbit et al.[26] proposed a fair queueing memory controller scheduling algorithm targeted for CMP systems. The controller allocates memory bandwidth to each thread based on the threads memory utilization. Excess bandwidth is then distributed across threads that have consumed less bandwidth in the past cycle. This guarantees that any malicious thread does not succeed in a denial of service attack on the machine.

2.2. Power Management

System growth in combination with device trends together have led to more and more devices being squeezed together on smaller and smaller areas [1]. This, in combination with speeds of operation etc., has resulted in growing power consumption. To effectively tackle this power consumption, most system components, including processors, disks and memory are capable of switching into low-power modes.

In the case of the memory system, power modes are available in nearly all DRAMs e.g. RDRAM, SDRAM, DDR/2. In DRAMs, a large portion of the power is drawn by the I/O circuitry, PLLs, on-chip registers. The low power modes disable this circuitry. Inter-node transitions take non-zero time, with the transition from low power modes to high power modes taking longer than transitions from high power modes to low power ones.

Rambus' RDRAM, for instance, supports four operating modes, in order of decreasing power consumption, active, standby, nap and idle. The granularity of control for power is at the level of an individual device or chip. The SDRAM family also offers power modes which are activated at the rank level. DDR2 SDRAM has a low power state- power down that is reached by disabling the clock enable (CKE) input. Depending on whether banks are precharged or active, the power-down savings are different. Additional power consumption determining factors are DIMM activity, such as bank activation, read and write operations, refresh etc.

Delaluz et al [27] also examined how to control DRAM power consumption for an RDRAM system. They examined how the compiler could insert directives to transition the DRAM into the appropriate power state based on profiling information. They also examined how to reorder array accesses and how to cluster arrays with similar access patterns together to reduce power consumption. They studied some hardware-based techniques that were threshold monitoring or history based techniques and found that these performed better because compiler-based techniques tended to be more pessimistic and lacked the detailed runtime information.

In a follow-up paper [28], they examined how operating system directed power management of the memory system could be beneficial. They observed that the OS can keep track of which pages are required by a process, and enable the associated modules prior to its scheduling, while disabling the idle modules. Power savings using this technique did not scale well with the number of modules, because of the uniform distribution of a process' pages across multiple modules. As the number of active threads increased, the returns also diminished.

Lebeck et al [29] examined how software and hardware techniques can be used to reduce power consumption in the memory system. They used both execution-driven and trace-driven simulations to quantify the impact of both hardware-only and software/hardware schemes for a system using RDRAM memory. The hardware based schemes included static schemes which transition the DRAM from one state to another after certain fixed threshold of time has passed, anddynamic schemes which examined distance between accesses to the same device before making a transitional decision. They studied how page allocation could be combined with the above schemes to improve energy reduction. Page allocation was such that the entire applications' data was allocated to one device at a time. Based on access patterns pages were migrated to the same chip, thereby allowing more devices to be powered down.

Huang et al [30] studied how a virtual memory manager could be used lower DRAM power consumption for a DRDRAM based system. They modified a virtual memory manager in Linux to perform page allocation in a more power efficient manner. The page allocation strategy used was similar to the sequential first-touch scheme used by Lebeck et al [29], but is enhanced to take into account DLL loading and shared pages. Unlike the earlier scheme, the operating system would issue instructions to the memory controller to activate a process' pages prior to its execution. The remainder of the DRAM sub system is kept in Nap mode. In a follow-up paper [31], they studied the impact of using power-aware virtual memory in a server with DDR based memory. Co-operative schemes perform marginally better than hardware-only schemes but use significantly lower resources to track page usage.

2.3. Commercial Memory Controllers

The 21174 memory controller[32], which was designed for the 21164 and 21164PC Alpha workstations[33], was an SDRAM based memory controller. This controller represented the transition from the use of asynchronous DRAM architectures to synchronous DRAM architectures. The goals of the design were to eliminate the latency incurred due to overheads like having to cross multiple chip domains. To do this, they used a novel memory sub-system design where the CPU was directly connected to the DRAM data bus, but the addressing and control was managed by the memory controller. The controller was designed for an open page system, and had a built-in 4-bit predictor per bank, which was used to determine whether the next access will be a hit or a miss. The prediction for a given predictor state was configured using a 16-bit software controlled register. They noted that the performance improvement by using this predictor is substantial for a few applications.

The Intel 870[34] is a memory controller for the Itanium. It can support up to 4 channels each with 8 DDR ranks. The memory controller chip can be connected to 4 processors simultaneously. It has an on-chip scalability port that enables it to be connected to an additional 12 processors. The chipset supports memory access re-ordering policies which focus on taking advantage of row locality and read/write re-ordering to avoid the impact of bus turn around times. The chipset also has its own read caches that act as prefect buffers for controller level pre-fetching. Being a multi-processor memory controller, it has support for directory level cache coherence. Several chipsets can be connected via the scalability port to form a network of 16-way processor system. Communication on this network is high-speed serial packet based communication.

The Intel front-side bus architecture has the processor communicating to the Northbridge chipset and cores via a fast, wide, shared bus. The northbridge chip, which was mainly the off-chip memory controller and cache coherence controller, is connected to the I/O controller, the AGP and the memory channels. With the trends towards increased integration, Intel first moved the graphics controller onto the chip-set[35]. More recently, the Intel 5000 series memory controller,(code-named *Blackford*), that is designed for dual-core and quad-core chips, takes this integration process further by moving the PCI Express controller onto the chipset [36]. The Blackford chipset supports 2 logical channels of FBDIMM memory (4 physical channels), that are referred to as “branches”. The chipset supports interleaving of cachelines across channels, ranks and banks. To provide increased RAS (Reliability, Availability and Serviceability), the memory is stored with ECC and the memory controller supports scrubbing i.e. periodically reading back memory and checking that it is correct. Both the PCI-express and FBDIMM channel are protected by CRC due to the higher transfer rates.

The increased integration of platform level components has resulted in the moving of the memory controller on-chip for both IBM’s Power 5 [37] and AMD Opteron processors [38, 39]. Both these chips support a dual-channel, 16-byte memory channel interface and reduce memory latency by eliminating a chip domain crossing. In the past, on-chip memory controllers have been built for the Sun Sparc 5, which used a simple 1 level caching hierarchy and an on-chip memory controller to reduce memory access overheads. Intel is expected to follow this trend with the *Nehelam* processor.

2.4. Processor-in-Memory Architectures

Traditionally complex OOO processors have been built to hide the memory latency. These processors use sophisticated techniques such as out-of-order execution and speculation to hide this latency. One requires large memories to keep these complex OOO processors busy. As the memory hierarchy gets more complex, the distance between the CPU and memory increases. Saulsbury et al.[40] proposed moving away from CPU-centric design to reduce the impact of the memory wall. They proposed bringing the processor and memory closer by moving the processor onto the DRAM chip.

The Berkeley IRAM[41, 42, 43, 44] project studied how to merge the processor and DRAM onto the same chip. They demonstrated how this could improve memory access latency, available bandwidth to the processor, overall energy efficiency and cost savings. Memory latency was reduced by redesigning the memory and allowing the processor to get data from accesses to rows which are closer to the processor earlier than those which were further away. This is unlike what is done in conventional DRAM chips. Energy reductions are achieved at comparable performance due to the lower cost of a DRAM access as compared to the an SRAM access. Further, due to the larger density of DRAM, the number of off-chip accesses are reduced resulting in an additional energy savings [44]. System cost reductions are achieved by reducing the number of chips on a mother board.

Vector IRAM[45, 46, 47] is an architecture that combines vector processing and IRAM to meet the demands of multimedia processing, with high energy efficiency. The vector IRAM processor comprises of an in-order superscalar core with one level of cache, a eight pipeline vector execution unit and several banks of memory. Code written for this architecture had to be compiled by a vectorizing compiler[46, 48] which was designed to compile code such that it took advantage of the on-chip memory bandwidth.

Another approach has been the FlexRAM architecture[49, 50] which is implemented on Merged Logic DRAM chips. The architecture comprises of many simpler processing elements each with a DRAM bank. Each compute element is restricted to access its own DRAM bank and that of its immediate neighbors. A larger processor element on the chip manages the execution of tasks on the simpler compute elements and the communication between non-adjacent members. The FlexRAM chip in turn can be connected onto any commodity memory interconnect. Cache coherence is managed either by the programmer or by using a directory based shared memory controller[51]. Programming for this architecture is made easier by the use of a special language and compiler support to automatically layout the code across the different compute elements[52, 53].

Some of the issues with building logic on DRAM technology [43] is that the latter has been optimized for small size and low leakage rather than speed. Further, the number of layers available in the two fabrication processes differ. The packaging used in DRAM chips is designed to dissipate significantly lower power (on the order of Watts) than that used by processors (can dissipate on the order of tens of watts). Due to the merging of logic and DRAM on the same chip also increases testing time.

The third approach proposed has been to use active pages[54, 55, 56], a page-based model of computation that associates simple functions with each page of memory. Active Page architectures are different from the previous two proposals since they are used to enhance performance of the conventional processor-memory architecture and not replace them, making it easier to adapt. Using active pages does not require the memory interface to be changed. Active Page data is modified with conventional memory reads and writes; Active Page functions are invoked through memory-mapped writes. Synchronization is accomplished through user-defined memory locations. Finally, Active Pages can exploit large amounts of parallelism by being able to support simultaneous computations to each of the pages in memory.

An alternate approach to reduce the distance between the processor and DRAM is to use a stacked micro-architecture. Black et al.[57] proposed a 3D die-stacked micro-architecture, where the DRAM is stacked on the CPU, thereby reducing memory latency and increasing bandwidth. Further, they demonstrate that this is a more power efficient architecture since it reduces the off-chip bus lengths.

2.5. Split-Transaction Buses

Shared buses used in multi-processor architectures are either single-transaction or split-transaction buses[58]. A single-transaction bus, also known as a circuit switched bus, permits only a single operation at a time, keeping the bus unavailable during the period when a read request is being serviced by the memory. A split-transaction bus permits multiple transactions to be outstanding by splitting each read request into two parts. First, a read request is followed by a release of the bus. Later, when the memory is prepared to return the result, it again arbitrates for the bus, acquiring it just long enough to send the requested data to the processor.

The split-transaction bus may be further refined by whether or not results are returned in the same order that they are requested. An in-order split-transaction bus always requires that read requests be completed in the same order in which they are initiated, while an out-of-order split-transaction bus places no restriction on the ordering of read requests. Typically, split-transaction buses may have a limit on the number of concurrent outstanding requests permitted

The HP Runaway Bus [59] was a split-transaction, time-multiplexed bus that was used in one-way to four-way SMP systems. It significantly improved bus utilization over a single-transaction bus. The bus supported multiple outstanding split transactions from each bus module, predictive flow control, a pipelined arbitration scheme and a snoopy coherence protocol. The bus protocol used master IDs and transaction IDs to tag every transaction.

These were transmitted in parallel with the address and the returning data, thereby ensuring that tag transmission did not contribute to overall latency.

Bus protocol design is also an important area of study in system-on-chip designs. Several protocols have been proposed to handle this problem including the AMBA bus protocol [60], the CoreConnect bus architecture [61], the Open Core protocol standard [62] by industry and others in academia including the LOTTERY bus [63] and the SAMBA protocol[64]. All these protocols use split transaction buses to improve bus utilization. The Samba protocol [64] attempts to improve fabrication delays in a split transaction bus system by explicitly defining request phases and response phases. The latter can be used by any module which does not share the same bus segment as the arbitration winner.

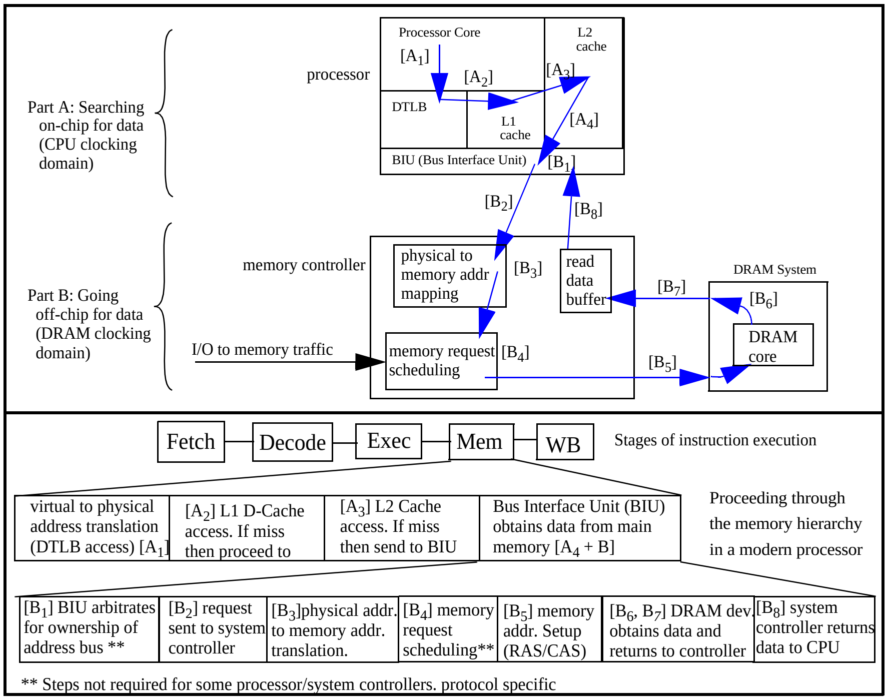
The concepts of using a split-transaction have also been explored in memory systems by Cuppu et al. [17, 65]. Cuppu et al. demonstrated how interleaving read and write data on the same bus can be used to improve bus utilization and overall system performance. Zhu et al [18] studied how splitting a memory request into multiple DRAM requests can be used to lower overall latency.

Chapter 3: Background

This chapter provides an overview of the workings of the memory system hierarchy and the operational parameters available to the designer. The chapter also delves into the main trends prevalent in the memory system industry. It describes the evolution of DRAM architecture, from asynchronous architectures used in the early 90’s to the current day synchronous dual data-rate devices and finally ends up with a detailed description of the FBDIMM memory architecture and protocol.

3.1. Memory Request Overview

Figure 3.1 illustrates the life of a memory request, right from its issue at the processor core, to its subsequent dispatch to the memory system and its completion in a uni-processor environment. Memory requests are issued by the core to handle load or store instructions and


Part A: Searching on-chip for data (CPU clocking domain)
processor
Processor Core
L2 cache
[A1]
[A2]
[A3]
DTLB
L1 cache
[A4]
BIU (Bus Interface Unit)
[B1]
[B2]
[B8]
Part B: Going off-chip for data (DRAM clocking domain)
memory controller
physical to memory addr mapping
[B3]
read data buffer
DRAM System
[B7]
[B6]
DRAM core
I/O to memory traffic
memory request scheduling
[B4]
[B5]
Fetch
Decode
Exec
Mem
WB
Stages of instruction execution
virtual to physical address translation (DTLB access) [A1]
[A2] L1 D-Cache access. If miss then proceed to
[A3] L2 Cache access. If miss then send to BIU
Bus Interface Unit (BIU) obtains data from main memory [A4 + B]
Proceeding through the memory hierarchy in a modern processor
[B1] BIU arbitrates for ownership of address bus **
[B2] request sent to system controller
[B3]physical addr. to memory addr. translation.
[B4] memory request scheduling**
[B5] memory addr. Setup (RAS/CAS)
[B6, B7] DRAM dev. obtains data and returns to controller
[B8] system controller returns data to CPU
** Steps not required for some processor/system controllers. protocol specific

igure 3.1. Abstract Illustration of Data being obtained from memory for a Load ıstruction [10].

Device configuration	**16 Meg x 16**
Configuration	4 M x 16 x 4 banks
row addressing	8K (A0 - A12)
bank addressing	4 (BA0, BA1)
column addressing	512 (A0- A8)

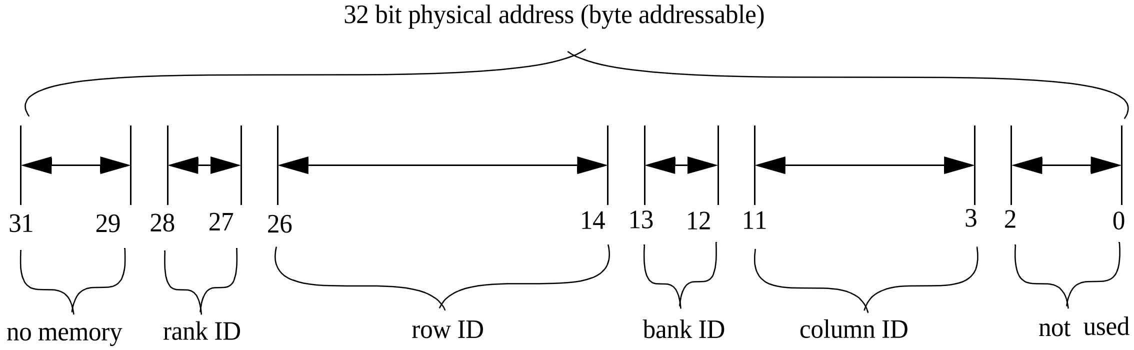


Figure 3.2. Address Mapping. The figure illustrates the address mapping scheme employed in a memory controller using DRAMs with the configuration in the table above. The device configuration influences how many bits are required to identify the address location. The system has one channel (hence no bits are allocated to the channel), four ranks of memory of the type specified in the table.

ALU instructions which use memory-based operands. If the request cannot be fulfilled from the processor's caches, a request is sent to memory controller.

Address Mapping: One of the first steps in processing a request is to do a mapping from the physical address space to the DRAM layout. The memory controller uses a fixed memory address mapping policy to do this mapping. An address location in DRAM is determined by a channel ID, a rank ID, a bank ID, a row ID and a column ID. The address mapping policy determines which bits of the physical address are used to determine the various DRAM specific IDs mentioned. Figure 3.2 illustrates how a address mapping policy determines the location of a given address in memory. Note, that since the system has only one channel, no address bits are allocated to the channel ID.

Address mapping policies are determined by the device configuration, system topology and row buffer management policies. The device configuration determines how many bits are allocated for the bank, row and column IDs. The system topology determines the layout of the ranks in the system, whether the ranks all exist on a single channel, multiple channels or multiple physical channels which are ganged to behave as a single logical channel.

Row buffer management policies are another important factor to consider when devising an address mapping policy. They influence which portion of the address bits to allocate to the various IDs. Typical row buffer management policies include the open page, closed page and auto page. An open page policy attempts to take advantage of the locality in a DRAM page, while a closed page policy is more common in systems where there is a little or no locality in the address stream. In an open page policy the row is opened and data is retained in the sense amps even after the transaction completes. The row is closed only when a new transaction which goes to another row in the same bank is scheduled. Thus, an address mapping policy for an open page system would try to capture this spatial locality. In a closed page system, the row is closed immediately upon the completion of a transaction and the banks are precharged. In such a system the address mapping policy will distribute adjacent addresses to different banks and ranks.

Memory Transaction Scheduling: Once the address translation is completed, the memory controller attempts to schedule the request to the DRAM. The scheduling policy of the controller and the state of the DRAM determines how long this process will take. For instance, the memory controller can select to re-order requests to obtain the most efficient

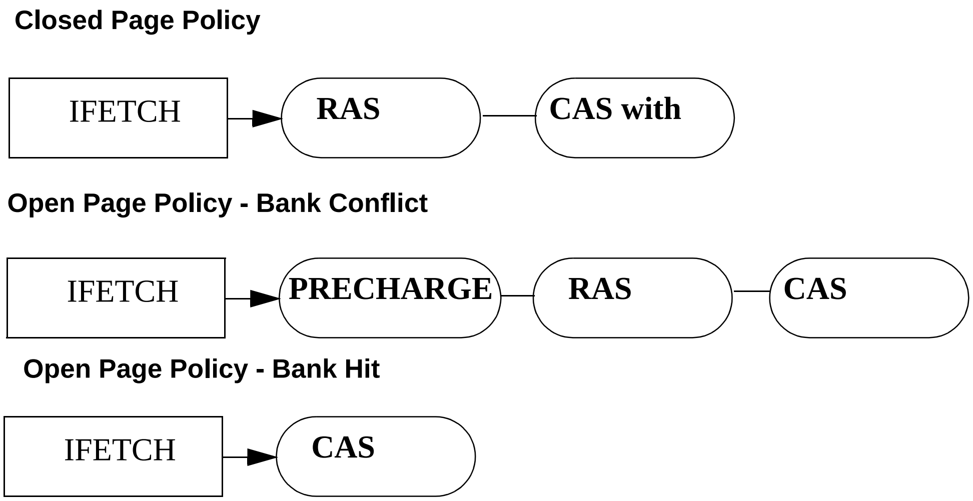


Figure 3.3. DRAM Commands associated with a transaction. The figure illustrates the resulting DRAM command stream for an instruction fetch (IFETCH) memory transaction. The commands scheduled are dependent on the state of the DRAM i.e. if the required row is open or not and the row buffer management policy in use.

bandwidth utilization. Based on the reordering policy used, priority can be given to reads over writes or to accesses that are mapped to an open row-buffer or to the oldest transaction etc. The transaction waits in the transaction queue waiting for its turn to be scheduled.

Scheduling to the DRAM: In this stage, the memory controller sends the associated DRAM commands to read or write the data. The transaction is broken down into a sequence of DRAM commands determined by the row buffer management policy and the state of the DRAM. Figure 3.3 demonstrates how these factors determine the exact DRAM command sequence used by an instruction fetch (IFETCH) transaction. In the closed page scheme, where the banks are by default precharged the memory controller has to first open the required row using a row activate command or RAS. The controller than issues a column access command or CAS to the DRAM which initiates a data burst from the DRAM. The controller restores the DRAM to its default state with a precharge command. In a system employing the open page policy, the relevant bank will be open but the open row may or may not be the row of interest. Depending on whether the relevant row is open (bank hit) or another row is open (bank conflict), the command sequence changes as illustrated.

Following the receipt of the relevant data, the memory controller sends this back to the processor, thereby completing the memory request.

3.2. DRAM Architectures over time

This section goes over the various types of DRAM and their features. Performance improvements in DRAM have been less influenced by advances in the speed of DRAM circuitry, than by higher level structural and interface changes. Bandwidth gains have been traditionally achieved by structural modifications to the interface. Recently, several DRAM manufactures have attempted to tackle the latency problem, by proposing radical changes to the underlying core architecture. Figure 3.4 illustrates the evolution in DRAMs, the changes between successive generations and which performance bottleneck the modifications attempt to address [10].

Conventional DRAM: forms the basis of modern day DRAMs with the addressing mechanism still in vogue, with minor changes. The address bus is multiplexed between row and column components. The multiplexed address bus uses two control signals of the row and column address strobe signals, RAS and CAS respectively which cause the DRAM to latch the address components. The row address causes a complete row in the memory array to propagate down the bit lines to the sense amps. The column address selects the appropriate data subset from the sense amps and causes it to be driven to the output pins.

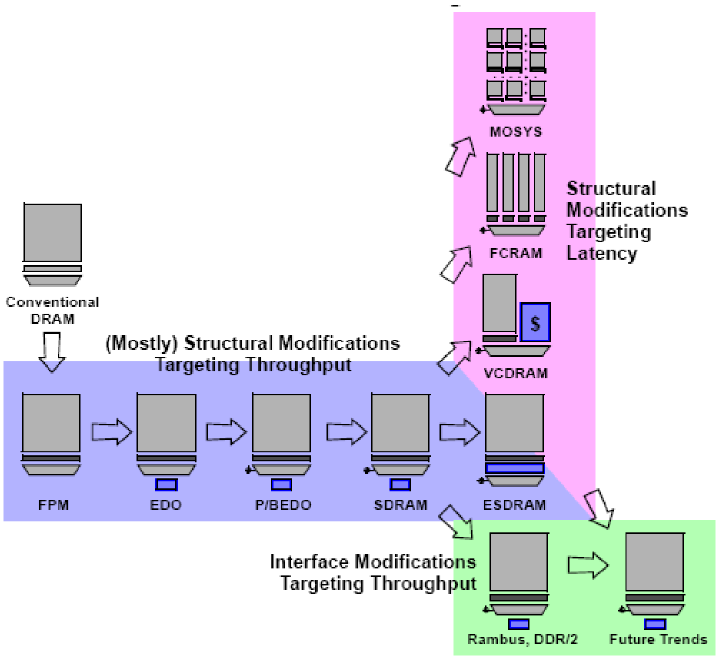


Figure 3.4. Evolution of the DRAM. The figure illustrates the trends in DRAM architectures from the basic DRAM to its modern day avatars. It shows the chief factors influencing the trends.

Fast Page Mode DRAM (FPM DRAM): Fast-Page Mode DRAM implements *page mode*, an improvement on conventional DRAM in which the row-address is held constant and data from multiple columns is read from the sense amplifiers. The data held in the sense amps form an open page that can be accessed relatively quickly. This speeds up successive accesses to the same row of the DRAM core.

Extended Data Out DRAM (EDO DRAM): Extended Data Out DRAM, sometimes referred to as hyper-page mode DRAM, adds a latch between the sense-amps and the output pins of the DRAM. This latch holds output pin state and permits the CAS to rapidly de-assert, allowing the memory array to begin precharging sooner. In addition, the latch in the output path also implies that the data on the outputs of the DRAM circuit remain valid longer into the next clock phase.

Synchronous DRAM (SDRAM) : Conventional, FPM, and EDO DRAM are controlled asynchronously by the processor or the memory controller; the memory latency is thus some fractional number of CPU clock cycles. An alternative is to make the DRAM interface synchronous such that the DRAM latches information to and from the controller based on a clock signal. SDRAM devices typically have a programmable register that to specify the burst length. The advantages include the elimination of the timing strobes and the availability of data from the DRAM each clock cycle. The underlying architecture of the SDRAM core is the same as in a conventional DRAM.

Enhanced Synchronous DRAM (ESDRAM): Enhanced Synchronous DRAM is an incremental modification to Synchronous DRAM that parallels the differences between FPM and EDO DRAM. First, the internal timing parameters of the ESDRAM core are faster than SDRAM. Second, SRAM row-caches have been added at the sense-amps of each bank. These caches provide the kind of improved intra-row performance observed with EDO DRAM, allowing requests to the last accessed row to be satisfied even when subsequent refreshes, precharges, or activates are taking place.

Dual Data Rate Memory (DDR): DDR has the same basic architecture as SDRAM. To achieve higher bandwidth rates of 200 to 400 Mbps, data is transferred at both edges of the clock. The higher clock speeds are achieved by multi-plexing the I/O buffers and not increasing the core speeds. A source-synchronous signal, DQS is added to enable data transfer. The DQS is generated by the component sending the data i.e. by the memory controller during a write and the DIMM during a read. The DQS is edge aligned for read data and centre aligned for write data.

Dual Data Rate Memory (DDR2): This is the next generation DDR DRAM with a few significant differences. DDR2 achieves higher data-rates lying between 400 to 667 Mbps. The number of DRAM banks is four for all devices except those with capacities greater than 1Gb. The uni-directional DQS signal has been replaced with a differential data strobe. DDR2 also introduces the concept of posted-CAS i.e. the memory controller can issue a RAS and CAS in subsequent cycles. The DRAM processes the CAS only after a fixed programmable period also known as the additive latency. The core operating voltage of DDR2 has been significantly lowered thereby lowering overall power consumption. Finally DDR2 adds on-die termination (ODT) to the data I/O pins. This feature is controlled by the ODT pin and consumes additional power when activated. Typically, on-die termination is only enabled to terminate write data to the DRAM or to terminate read data from a different DRAM.

Virtual Channel SDRAM (VCSDRAM): Virtual Channel SDRAM, designed by NEC, also contains SRAM caches. But the caches are not really buffers for the sense amps like ESDRAM. VC SDRAM contains 16 virtual channels, or 16 1 KB SRAM caches. While the ESDRAM module takes care of the "caching" internally, the VC SDRAM caches are managed by the chipset. This results in two important consequences. First, VC SDRAM will only work properly when paired with a chipset that supports it. Additionally, the performance of VC SDRAM will depend tremendously on the quality of the chipset's implementation.

Fast Cycle RAM (FCRAM) : FCRAM (Fast Cycle RAM) was developed by Fujitsu which seeks to lower latency by changing the DRAM core. The process includes core segmentation and pipeline operations. This new structure has some advantages including the ability to send Row and Column information at the same time, as opposed to the standard sequential operation. FCRAM uses a standard DDR interface.FCRAM targets high-performance network systems and other high-end applications that require high-speed large capacity low-power memories.

3.2.1 Serial Memory Protocols

In the late 90s, Rambus released the first high-speed serial bus based memory system protocol[7, 5]. The proposal, which was radical for that time, was largely an overhaul of the conventional memory system interface. Initial Rambus Base Common Interface replaced the wide data bus with a bi-directional byte-wide data bus. Two additional bus control bus signals and the data bus were used to transport command information. DRDRAM increased the data bus interface to 16 bytes wide and added on a dedicated row and command bus. Higher data-rates were achieved by sending data every clock edge and reflection was reduced by using termination, both features which were seen in DDR systems. The architecture also incorporated an interesting clocking scheme, where the clock travelled from the memory controller to the DRAM chips and then looped back and returned on a clock return path to the memory controller. Data to the chips was synchronized with the outward bound clock while data being sent back to the controller was synchronized with the clock on its return path.

To support the narrow, high-speed interface RDRAM chips had decoding logic, DLLs built on. These increased both the power consumption and manufacturing costs of the RDRAM chips. DRDRAM which was the second generation Rambus DRAM standard, had a pipe-lined micro-architecture with different pipeline stages dedicated to row decoding, command transport, data transfer and moving data into and out of a write buffer. To utilize the available pin bandwidth in DRDRAM devices, the DRDRAM chips typically had at least 16 DRAM banks, each with 1 Kilobyte data pages. The number of banks are higher than that seen in a conventional SDRAM or DDR device.

Although selected to be the memory platform for Intel’s processors, DRDRAM never ended up becoming the default industry standard, in part due to the larger costs associated with manufacturing and licensing. Further, DRDRAM also were demonstrated to have higher latency characteristics[6] as well as higher power consumption. DRDRAM was used in gaming systems like the Sony Play Station 2 where high bandwidth was a necessity.

XDR is the next generation Rambus serial memory protocol which has an octal data-rate of 3.2 Gbps. This is achieved by transmitting 8 data bits every clock cycle. Unlike past offerings from Rambus, XDR uses differential signalling, also known as Rambus Differential Signalling, on the bus. XDR does away with path length matching by using flexphase technology. Flexphase refers to the per bit deskewing control used in XDR. The burst length of an XDR system,16, is significantly longer than current DDRx systems. Most Rambus solutions described above are all part of solutions to deal with providing higher data-rates by utilizing high-speed serial narrow buses.

3.3. Memory System Trends

The memory system over a decade has been subject to several generic trends, higher bandwidths achieved by using wide, fast buses, and increasing DRAM capacity per chip. These trends have till date supported the memory requirements of both desktop and server systems. Further scaling the conventional DRAM interface to meet the increased bandwidth and capacity demands are less successful due to the inherent limitations placed by wide high-speed buses. In the following section, we examine the current trends in the memory system industry, future growth expectations, and how the combinations has resulted in the proposal for an alternate memory interface.

3.3.1 Bandwidth

The Von Neumann architecture [15], also known as the 'stored-program concept' is the universally accepted model of computing. The separation of the CPU and the memory contributes to what is known as the Von Neumann bottleneck, a term coined by John Backus in his 1977 ACM Turing award lecture. The bottleneck refers to the data transfer rate of the bus between the CPU and the memory. This bus has a far lower capacity than the memory connected to it, thereby becoming a bottleneck in the transfer of data from the memory to the processor.

According to Backus:

> *"Surely there must be a less primitive way of making big changes in the store than by pushing vast numbers of words back and forth through the von Neumann bottleneck. Not only is this tube a literal bottleneck for the data traffic of a problem, but, more importantly, it is an intellectual bottleneck that has kept us tied to word-at-a-time thinking instead of encouraging us to think in terms of the larger conceptual units of the task at hand. Thus programming is basically planning and detailing the enormous traffic of words through the von Neumann bottleneck, and much of that traffic concerns not significant data itself, but where to find it."*

This von Neumann bottleneck has been aggravated by growing CPU speeds and increasing memory capacity. The slower memory speed which increases access time to memory worsens this bottleneck. The latency aspect which has been the focus of computer architecture research will be discussed later. The initial efforts to deal with this bottleneck has been the usage of caches, which are located closer to the processor and hold the operating set.

Several factors have contributed to the increasing memory bandwidth requirement demands. These include the system level trends like increasing processor speeds, growing use of speculative techniques, and increasing number of cores/threads on the die. Another aspect driving increasing memory bandwidth requirements has been application driven trends, shown in figure 3.5 [18].

Bandwidth demands were initially met by increasing the width of the memory data-bus, the data-rate and simultaneously the internal prefetch width. As can be seen in figure 3.6 the data bus width has increased from 8 bits to its current day value of 64 bits, by doubling nearly every 3 processor generations.

The other technique used to increase bandwidth has been to increase the speed of the front-side bus. As can be seen in figure 3.6, the data bus speed has been doubling every 2 years. The figure also shows that DRAM core speed has been increasing at a far lower rate than the increases in pin-bandwidth. The recent data-rate increases have been largely fuelled by the use of internal pre-fetching. For example, in the SDRAM the core and the

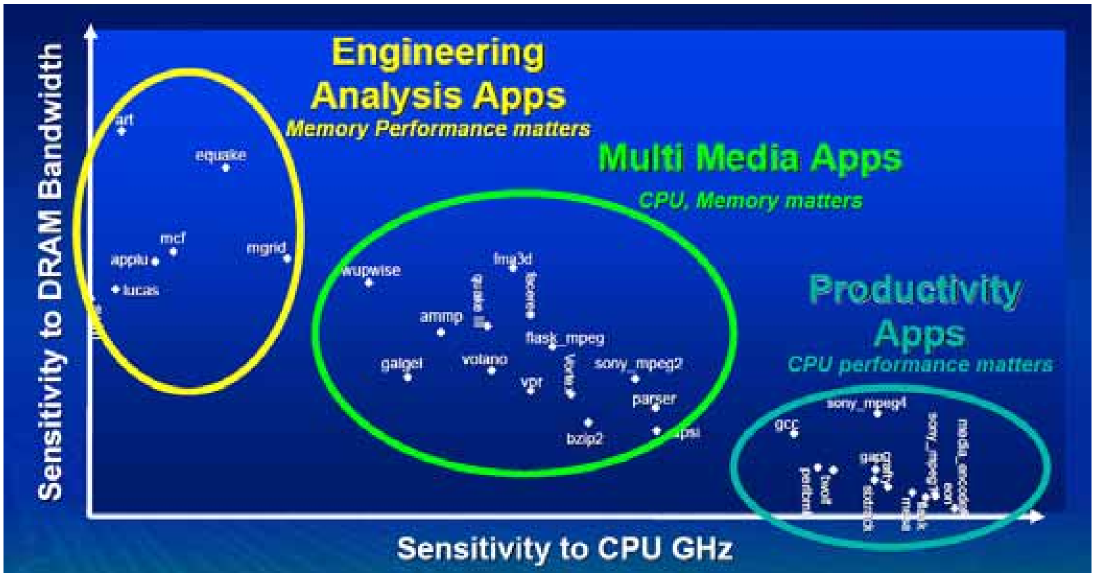


Figure 3.5. Application sensitivity to Bandwidth and CPU frequency. The figure shows a classification of commonly used benchmarks and applications, to CPU frequency and memory bandwidth. *Source Intel.*

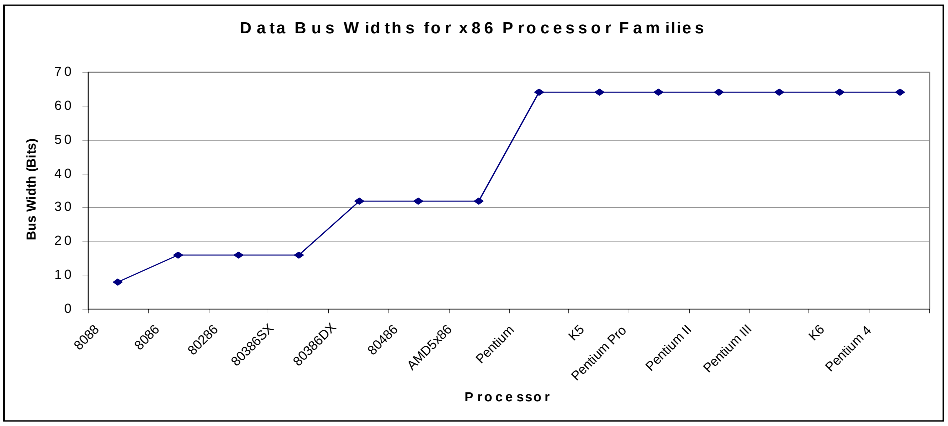


Figure 3.6. Memory Data Bus Widths for Different Processors. The figure shows the increasing processor data bus width with succeeding generations of x86 processors. The current data bus width is 64 bits.

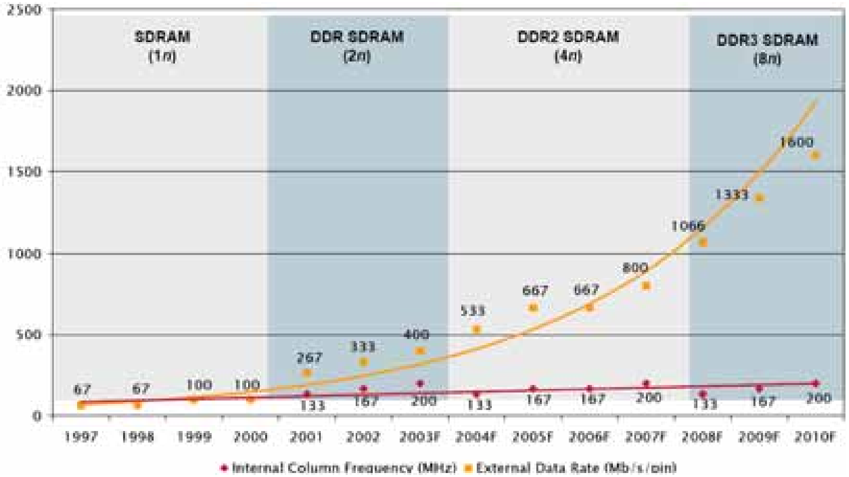


Figure 3.7. DRAM Data Rate Trends. The figure shows the trends in DRAM data rate, internal core speed. The speed-rate increases between generations of DRAM is achieved by increasing the internal prefetch.

bus are running at the same rate, with each output buffer releasing a bit every clock cycle. For DDR, the core speed is half the bus speed. In this case, two bits are transferred to the output buffer and released in a time multiplexed fashion at both the falling and rising edge of the clock. With DDR2 and DDR3, the prefetch rate is increased by 4 or 8, allowing the core to run at a significantly lower speed than the actual interface.

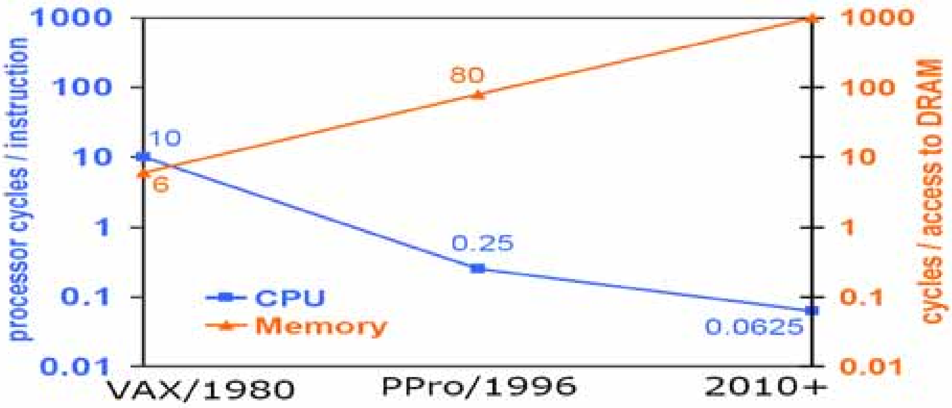


Figure 3.8. Memory/Processor Speed Gap. The graph depicts the "memory wall" in terms of the latency to DRAM in CPU cycles, and the increasing ability of the processor to process and commit more and more instructions per cycle.

Increasing the data-rate is difficult as the parallel bus architecture begins to show limitations at speeds touching gigabits-per-second. Noise and cross-talk at higher speeds reduce signal integrity and make bus synchronization non-trivial. The energy required to drive a wide bus is high, exacerbating ground bounce, noise problems and increasing the power dissipation. At higher frequencies, bus turn-around time can no longer be hidden. Hence, after years of ever-widening bus widths, narrow serialized buses have emerged as a natural choice.

3.3.2 Latency

The von Neumann bottleneck is further aggravated by the growing separation of speeds between the memory and the processor. CPU speeds double roughly every 18 months, while DRAM speed i.e. speed of the core increases at a more modest rate of roughly 7% only. This difference in speed contributes to the growing latency of a memory request, in terms of CPU cycles (Figure 3.8) [1]. This resulting latency increase has been termed the "memory wall"[17].

Techniques to tolerate, reduce and hide this latency have been developed. These techniques include lock-up free caches [11], hardware and software pre-fetching [3, 4], speculative execution and multi-threading. These techniques trade-off reducing tolerating memory latency while increasing memory bandwidth [2]. A few alternate DRAM technologies like FCRAM, VCSDRAM have also been proposed to tackle the latency problem. These techniques though have been used only high-end systems where the additional costs are justified.

3.3.3 Capacity

The desktop requirements for RAM (based on the minimum requirements to boot a version of Microsoft Windows) has roughly doubled every two years from around 4MB in 1993 to 256MB in 2004. The minimum requirements for Windows are at least half of what is required to run a system without frequent swapping of pages to disk. Server DRAM requirements has also followed a similar trend, as shown in the figure 3.9 [14].

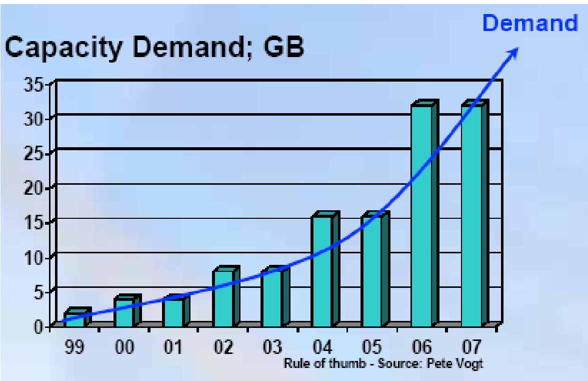


Figure 3.9. Server Capacity Demand. The figure shows the trend in RAM requirements for servers and demonstrates how the installed capacity has tracked this demand.

The DRAM chip density growth has slowed down from roughly 4X every 3 years to roughly 2X every three years (figure 3.10) [9]. The most recent introduction has been the 1Gb chip on 90 nanometer technology by Samsung. DRAM chips are combined to dual-inline memory modules (DIMMs). DIMM capacity is a function of both the number of DRAM chips used and the capacity of the chips. Dimms that feature 36 Drams, also called “high density” or stacked DIMMs are used in server systems.

The capacity per DIMM has been increasing due to improved DRAM density over time, unfortunately, total capacity per channel (the number of DIMMs per channel multiplied by the capacity per DIMM) has stayed flat at best and has even decreased in certain instances. The multi-drop bus in vogue today has each DIMM directly connected to the channel. To reduce the impedance mismatches, the number of DIMMS in a channel are limited, thereby limiting the DRAM capacity of a system. The number of loads/channel has been decreasing from SDRAM/DDR,8, to DDR2,4 and further to DDR3 i.e. 1-2 (figure 3.11).

An alternate solution to increasing capacity would be to increase the number of channels. This is not possible, because of the width of the channel. The 240-pin wide interface makes routing already a complex and expensive task due to path-length matching con-

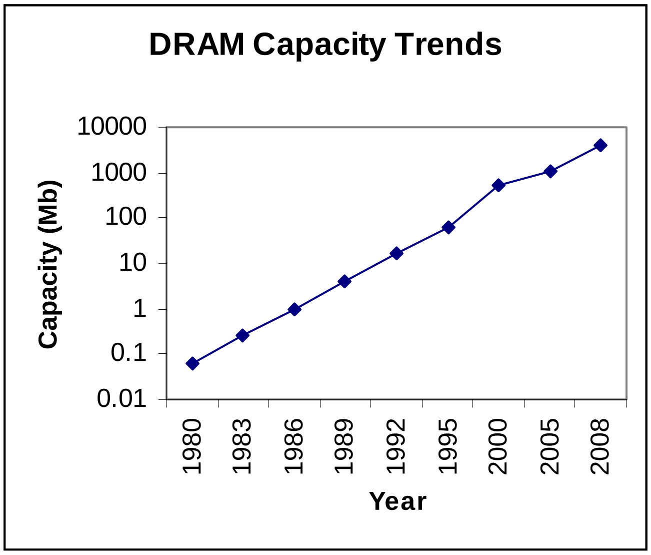
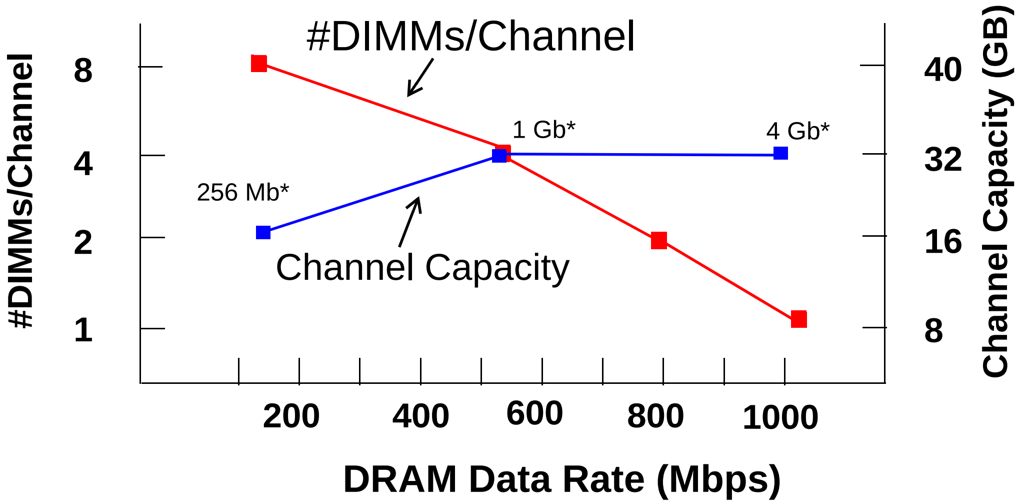


Figure 3.10. DRAM Capacity Trends. The graph shows the increase in DRAM bit density over the years. The initial trend of roughly 4X every 4 years has slowed down to 2X every 4 years.



*DRAM Chip Bit Density

Figure 3.11. DIMM Capacity/Channel with increasing data-rates. The figure shows how the number of DIMMs/channel has been steadily decreasing with increasing data rate. Source [8].

straints. The resulting serpentine routing also occupies significant board area. Increasing the number of channels, would significantly increase the cost of the motherboard as well as the complexity of its design.

3.4. FBDIMM Overview

As DRAM technology has advanced, the channel speed has improved at the expense of channel capacity; consequently channel capacity has dropped from 8 DIMMs to a single DIMM per channel. This is a significant limitation—for a server designer, it is critical, as servers typically depend on memory capacity for their performance. There is an obvious dilemma: future designers would like both increased channel capacity and increased data rates—but how can one provide both?

The trend is clear: to improve bandwidth, one must reduce the number of impedance discontinuities on the bus. Graphics subsystem designers have known this for a long time—for every DRAM generation, graphics cards use the same DRAM technology as is found in commodity DIMMs, but operate their DRAMs at significantly higher speeds by avoiding multi-drop connections.

The relation between a modular organization and the graphics-card organization is shown in Figure 3.12, which depicts a multi-drop DRAM bus next to a DRAM bus organization typical of graphics cards. The graphics card uses the same DRAM technology as in the multi-drop organization, but it uses point-to-point soldered connections between the DRAM and memory controller to achieve higher speeds. How can one exploit this to increase data rates without sacrificing channel capacity? One solution is to redefine one's terms—to call the graphic-card arrangement a “DIMM” in one's future system. This is shown on the right-hand side of figure 3.12 and is precisely what is happening in FB-DIMM: a slave memory controller has been added onto each DIMM, and all connections in the system are point-to-point. The channel connecting the master memory controller to the DIMM-level memory controllers (called *Advanced Memory Buffers*, or *AMBs*) is very narrow and very fast. Because each DIMM-to-DIMM connection is a point-to-point connection, a channel becomes a *de facto* multi-hop store & forward network. The FB-DIMM architecture limits the channel length to 8 DIMMs, and the width of the inter-module bus is narrow enough to require roughly one third the pins as a traditional organization. The result is that an FB-DIMM organization can handle roughly 24 times the storage capacity as a single-DIMM DDR3-based system, without sacrificing any bandwidth and even leaving headroom for increased intra-module bandwidth.

The AMB behaves like a pass-through switch, directly forwarding requests it receives from the controller to successive DIMMs and forwarding frames from southerly

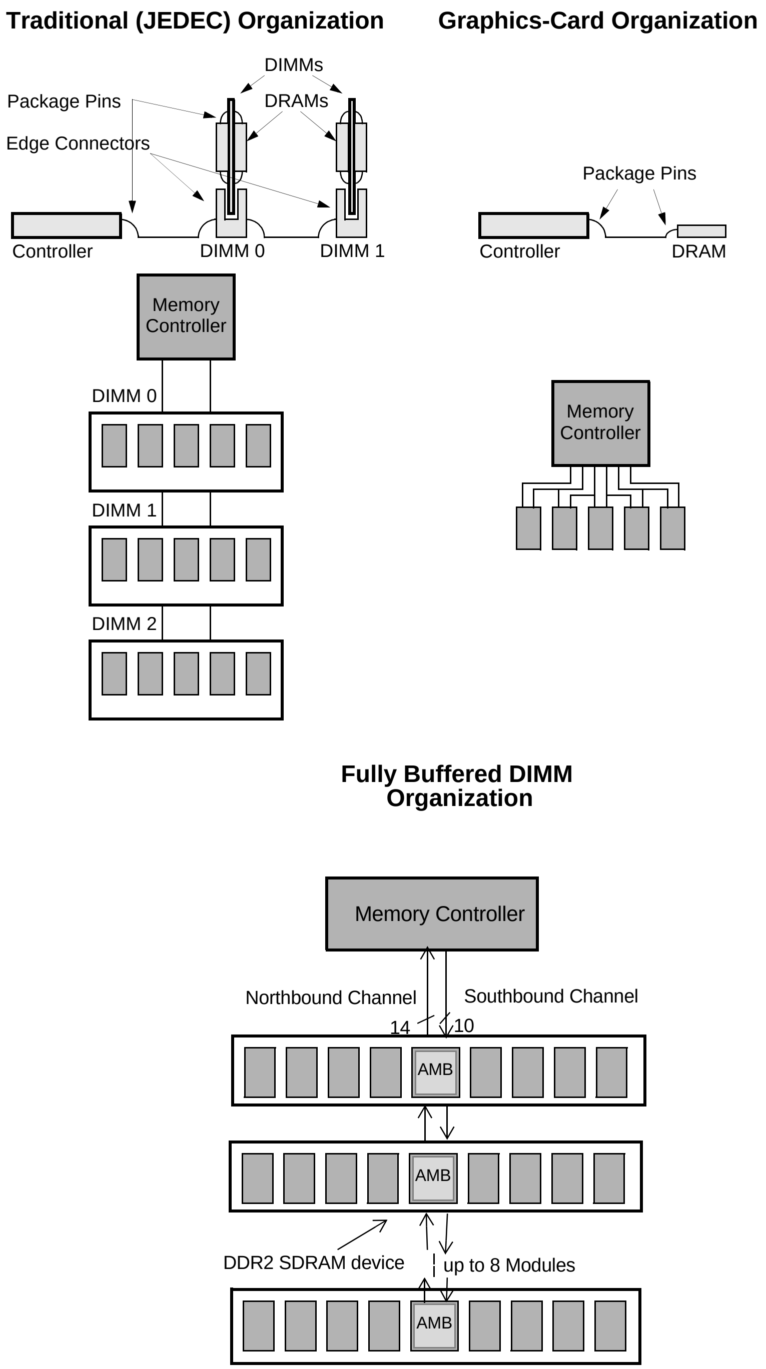


Figure 3.12. Motivation for the design of FBDIMM. The first two organizations compare a modular wide-bus and memory organization of a graphics card. Above each design is its side-profile, indicating potential impedance mismatches (sources of reflections). The organization on the far right shows how the FB-DIMM takes the graphics-card organization as its *de facto* DIMM. In the FB-DIMM organization, there are no multi-drop busses; DIMM-to-DIMM connections are point-to-point. The memory controller is connected to the nearest AMB, via two uni-directional links. The AMB is in turn connected to its southern neighbor via the same two links.

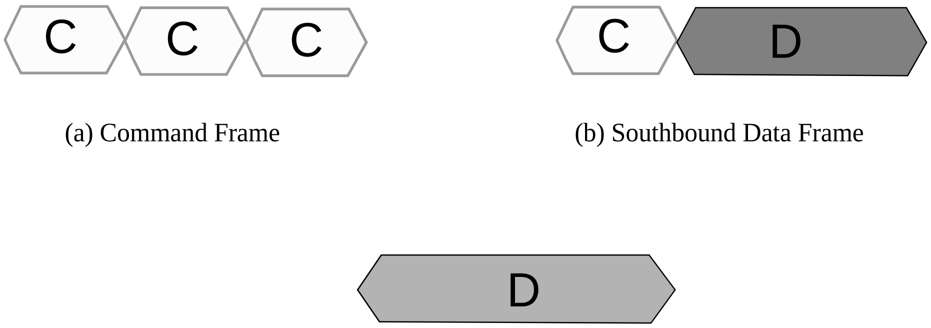
DIMMs to northerly DIMMs or the controller. All frames are processed to determine whether the data and commands are for the local DIMM.

3.4.1 FBDIMM protocol

The FBDIMM system uses a serial packet-based protocol to communicate between the memory controller and the DIMMs. Frames may contain data and/or commands. Commands include DRAM commands such as row activate (RAS), column read (CAS), refresh (REF) and so on, as well as channel commands such as write to configuration registers, synchronization commands etc. Frame scheduling is performed by the memory controller. The AMB only converts the serial protocol to DDRx based commands without implementing any scheduling functionality.

The AMB is connected to the memory controller and/or adjacent DIMMs via serial uni-directional links - the southbound channel which transmits both data and commands and the northbound channel which transmits data and status information. The southbound and northbound data paths are respectively 10 bits and 14 bits wide. The channel is dual-edge clocked i.e. transmission of data is done on both clock edges. The different bus widths for the northbound and southbound channel ensure that a FBDIMM system has twice the read bandwidth. The read bandwidth of a FBDIMM system matches the bandwidth of a DDRx system while the write bandwidth available is one half of a DDRx system making the total bandwidth available 1.5 times that in a DDRx system.

The FBDIMM channel clock operates at 6 times the speed of the DIMM clock i.e. the link speed is 4.2 Gbps for a DDRx system operating at 667 Mbps. Frames on the north and south bound channel require 12 transfers (6 FBDIMM channel clock cycles) for transmis-



(a) Command Frame

(b) Southbound Data Frame

(c) Northbound Data Frame

Figure 3.13. FBDIMM Frames. The figure illustrates the various types of FBDIMM frames which are used to transfer commands and data in the FBDIMM system.

sion. This 6:1 ratio ensures that the FBDIMM frame rate matches the DRAM command clock rate.

Southbound frames comprise both data and commands and are 120 bits long; data-only northbound frames are 168 bits long. In addition to the data and command information, the frames carry header information and frame CRC checksum used to check for transmission errors.

The types of frames defined by the protocol are [12] and illustrated in the figure 3.13

Command Frame: comprises up to three commands (figure 3.13a). These are sent to separate DIMMs on the southbound channel. In the absence of available commands to occupy a complete frame, no-ops are used to pad the frame. Frames are protected by 22-bit CRC when the channel is fully operation or 14 bit-CRC when the channel experiences a single bit lane failure.

Command + Write Data Frame: carries 72 bits of write data and one command on the southbound channel (figure 3.13b). Multiple such frames are required to transmit an entire data block to the DIMM. The AMB of the addressed DIMM buffers the write data as required.

Data frame: comprises 144 bits of data or 16 bytes of data and 2 bytes of ECC information (figure 3.13c). Each frame is filled by the data transferred from the DRAM in two beats or a single DIMM clock cycle. This is a northbound frame and is used to transfer read data.

The FBDIMM has two operational modes, the fixed latency mode and the variable latency mode. In fixed latency mode, the round-trip latency of frames is set to that of the furthest DIMM. Hence, systems with deeper channels have larger average latencies. In the variable latency mode, the round-trip latency from each DIMM is dependent on its distance from the memory controller.

Figure 3.14 shows the processing of a read transaction in an FBDIMM system. Initially a command frame is used to transmit a command that will perform row activation. The AMB translates the request and relays it to the DIMM. The memory controller schedules the CAS command in a following frame. The AMB relays the CAS command to the DRAM devices which burst the data back to the AMB. The AMB bundles two consecutive bursts of a data into a single northbound frame and transmits it to the memory controller. The AMB releases the data on the northbound link In this example, we assume a burst length of 4 corresponding to 2 FBDIMM data frames. Note that although the figures do not identify parameters like t_CAS, t_RCD and t_CWD [16, 13] the memory controller must ensure that they are met.

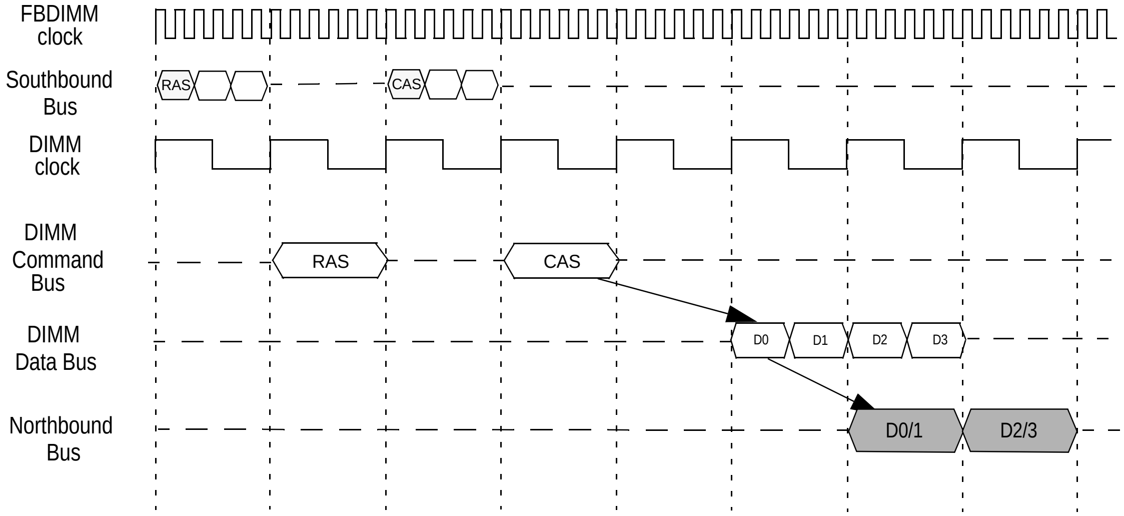
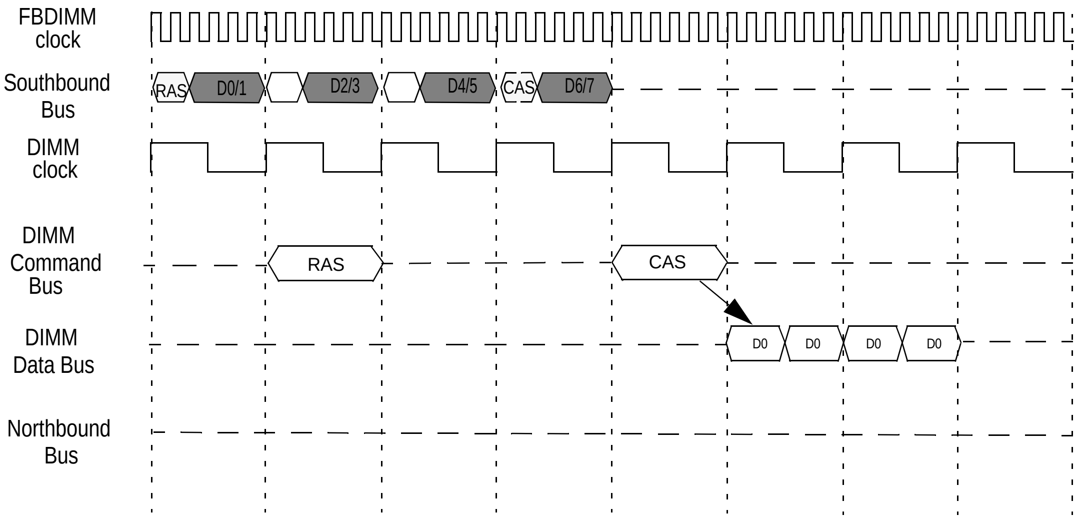


Figure 3.14. Read Transaction in FBDIMM system. The figure shows how a read transaction is performed in a FBDIMM system. The FBDIMM serial buses are clocked at 6 times the DIMM buses. Each FBDIMM frame on the southbound bus takes 6 FBDIMM clock periods to transmit. On the northbound bus a frame comprises of two DDRx data bursts.

Figure 3.15 shows the processing of a memory write transaction in a FBDIMM system. Due to the reduced south link bandwidth the write data requires double the number of frames than the read data. All read and write data are buffered in the AMB before being relayed on, indicated by the staggered timing of data on the respective buses. The CAS command is transmitted along with the last chunk of data



igure 3.15. Write Transaction in FBDIMM system. The figure shows how a write ansaction is performed in a FBDIMM system. The FBDIMM serial buses are clocked at 6 mes the DIMM buses. Each FBDIMM frame on the southbound bus takes 6 FBDIMM ock periods to transmit. A 32 byte cacheline takes 8 frames to be transmitted to the DIMM.

Chapter 4: Performance Evaluation

In this chapter, we study how the performance of an FBDIMM system varies with system configuration parameters. We also study how the performance of an FBDIMM system compares with its corresponding DDRx system. Our studies show that the relative performance of a FBDIMM system as compared to a DDRx system is a strong function of the bandwidth utilization of the workload. We found that memory controller optimizations employed in conventional memory systems are still relevant in FBDIMM systems. FBDIMM systems are limited by mainly the efficient utilization of the channel bandwidth available.

4.1. Methodology

This study uses DRAMsim [9], a stand-alone memory system simulator. DRAMsim is a detailed execution-driven simulator that models a number of memory protocols including DDRx systems and their corresponding FBDIMM systems. The simulator supports the variation of memory system parameters of interest including scheduling policies, memory configuration i.e. number of ranks and channels, address mapping policy etc.

4.1.1 Memory Controller Model

Figure 4.1 depicts the architecture of the memory controller simulated for our study and the mechanism used to drive the simulator for this study. The memory controller comprises of two equally sized transaction queues - the read transaction queue and write transaction queue, from which command and data frames are scheduled to the channel. Read transactions are moved to the Read Response Queue, once the associated DRAM commands are scheduled, where they wait for the read data to return. The memory controller

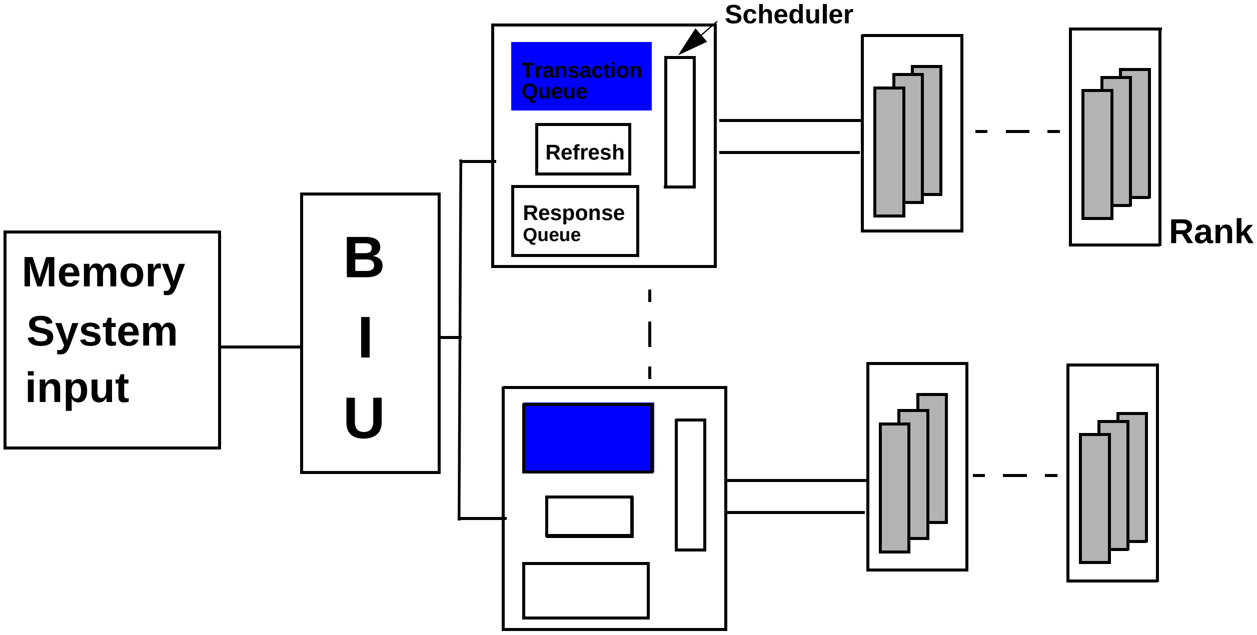


Figure 4.1. Memory Controller Architecture. .The figure shows the basic architecture of th memory controller modelled in DRAMsim and used fro these experiments. The figure also shows th methodology used to input transactions into the simulator. Memory requests are fed from either a fil created by a random address generator or from a memory request trace file obtained from CPU simulation infrastructures.

guarantees that read and write requests to identical DRAM locations are serialized. However, we do not model any additional constraints imposed by the use of particular consistency and coherency protocols.

We studied four different DRAM types: a conventional DDR2 system with a data-rate of 667Mbps, a DDR3 system with a data rate of 1333Mbps and their corresponding FBDIMM systems: an FBDIMM organization with DDR2 on the DIMM and another with DDR3 on the DIMM. FBDIMM modules are modelled using the parameters available for a 4GB module [5]; DDR2 device are modelled as 1Gb parts with 8 banks of memory [6]; DDR3 parameters were based on the proposed JEDEC standard[4]. The ratio of the CPU frequency to the corresponding DDRx data-rate was set to 3:1. The timing parameters used in this study are given in the table 4.1.

TABLE 4.1. Memory System Parameters

Parameter	**DDR2**	**DDR3**
Data-rate (Mbps)	667	1334
t_{RAS} (ns)	45	36
t_{RP} (ns)	15	12
t_{RC}(ns)	60	48
t_{RCD} (clock cycles)	5	8
t_{CAS} (clock cycles)	5	8
t_{RFC}(ns)	127.5	110
t_{FAW}(ns)	37.5	30
t_{RRD}(ns)	7.5	6
t_{RTP}(ns)	7.5	7.5
Transaction Queue Delay (ns)	16	16

Multi-programming workloads were generated by combining several SPEC 2000 benchmarks [3]. The workloads were combined to generate two main types of combinations, a memory intensive (MEM) and a mix of memory intensive and ILP intensive applications (MIX). The methodology used to identify whether a given SPEC benchmark is memory intensive or not and to create the workload mixes is based on those proposed in earlier SMT studies [1, 8]. SPEC benchmark traces were gathered using Alpha 21264 simulator sim-alpha[2], with a 1 MB last-level cache and an out-of-order core. The memory address traces collected have the time-stamp associated with every memory request. The BIU interface is setup such that requests are placed into the BIU at the time they arrive, as determined by the trace. The changes in timing introduced by the use of a different memory system configurations are incorporated into the trace timing behavior as well.

TABLE 4.2. Multi-programming Workloads

Workload	Benchmarks	Type
MIX-2a	mcf, gzip	MIX
MIX-2b	applu, wupwise	MIX
MEM-2a	mcf, gcc	MEM
MEM-2b	swim, art	MEM
MIX-4a	applu, wupwise, mgrid, ammp	MIX
MIX-4b	swim-gzip-art-vortex	MIX
MEM-4a	swim-lucas-mcf-applu	MEM
MEM-4b	mcf-applu-mgrid-art	MEM
MIX-8a	swim-gzip-lucas-vortex-applu-wupwise-mgrid-ammp	MIX
MIX-8b	swim-gzip-lucas-vortex, mcf, gcc, applu, gzip	MIX
MEM-8a	mcf-applu-mgrid-art-swim-lucas-swim-lucas	MEM
MEM-8b	mcf-applu-mgrid-art-swim-lucas-mcf-applu	MEM

Several configuration parameters that were varied in this study include:

Number of ranks and their configuration. FBDIMM systems support six channels of memory, each with up to eight DIMMs. This study expands that slightly, investigating a configuration space of 1-32 ranks, with ranks organized in 1-4 channels of 1-8 DIMMs each. Though many of these configurations are not relevant for DDRx systems, we study them to see the impact of a different bus topology on performance.We study configurations with one rank per DIMM. Hence, the phrase rank and DIMM interchangeably throughout this dissertation.

Row-buffer management policies . We study both open-page and closed-page systems. Note that the address mapping policy is selected so as to reduce bank conflicts in the system. The address mapping policies “sdram close page map” and “sdram hiperf map” [9] provide the required mapping for closed page and open page systems respectively.

Scheduling policies . Several DDRx DRAM command scheduling policies were studied. These include

- **Greedy** The memory controller issues a command as soon as it is ready to go. Priority is given to DRAM commands associated with older transactions.
- **Open Bank First (OBF)** is used only for open-page systems. Here priority is given to all transactions which are issued to a currently open bank.
- **Most pending** is the “most pending” scheduling policy proposed by Rixner [7], where priority is given to DRAM commands to banks which have the maximum number of transactions.
- **Least pending** is the “least pending” scheduling policy proposed by Rixner [7]. Commands to transactions to banks with the least number of outstanding transactions are given priority.
- **Read-Instruction-Fetch-First (RIFF)** prioritizes memory read transactions (data and instruction fetch) over memory write transactions.

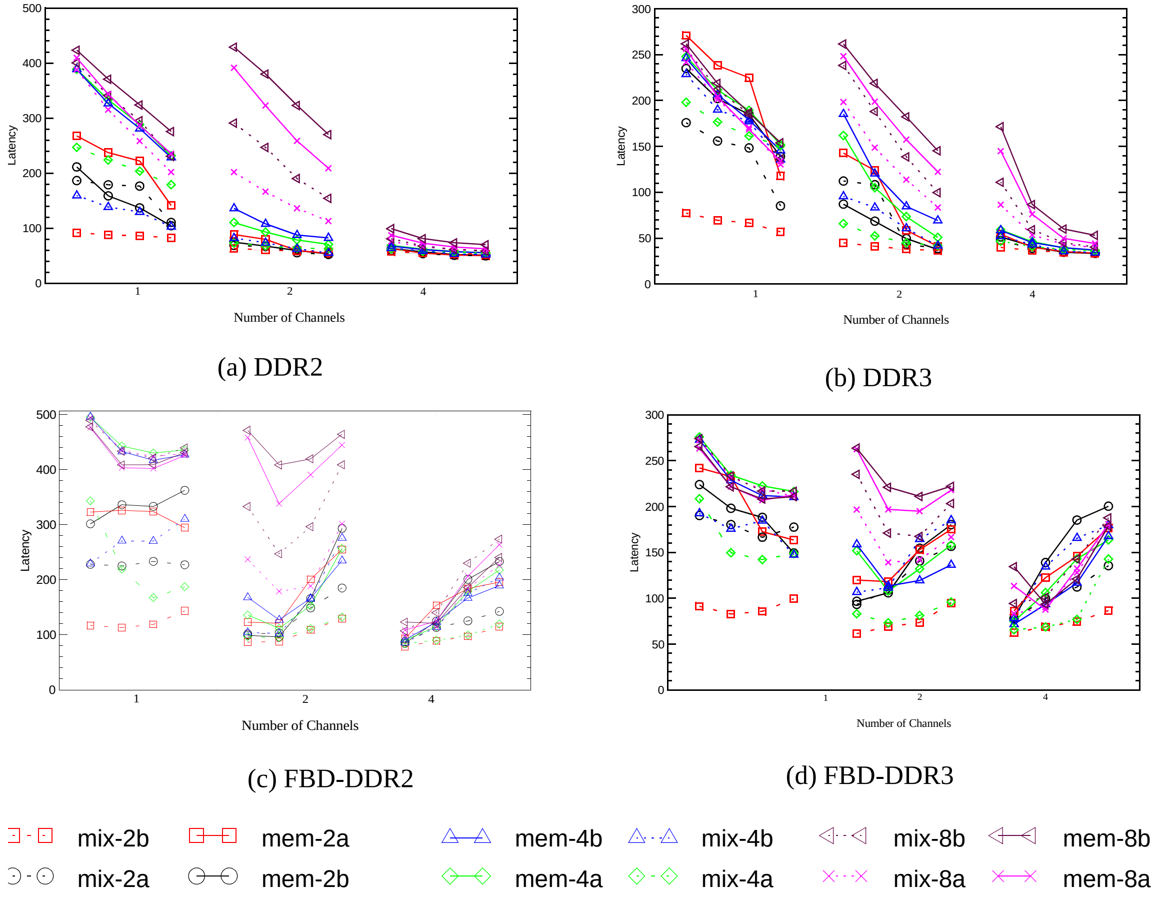
Note that we only study the scheduling policies OBF, least pending and most pending in the case of open page systems since these policies attempt to improve latency characteristics by exploiting row locality.

4.2. FBD vs. DDRx

In this section, we describe how the performance characteristics of the various workloads varied with a number memory controller configuration parameters. We also examine how memory sub-system performance i.e. latency and bandwidth changed when replacing a conventional DDRx system with its FBDIMM counterpart.

4.2.1 Application Latency and Bandwidth Characteristics

Figure 4.2 and 4.3 show the variation in average read latency with system configuration for all the workloads executing in a closed page system and in an open page system,



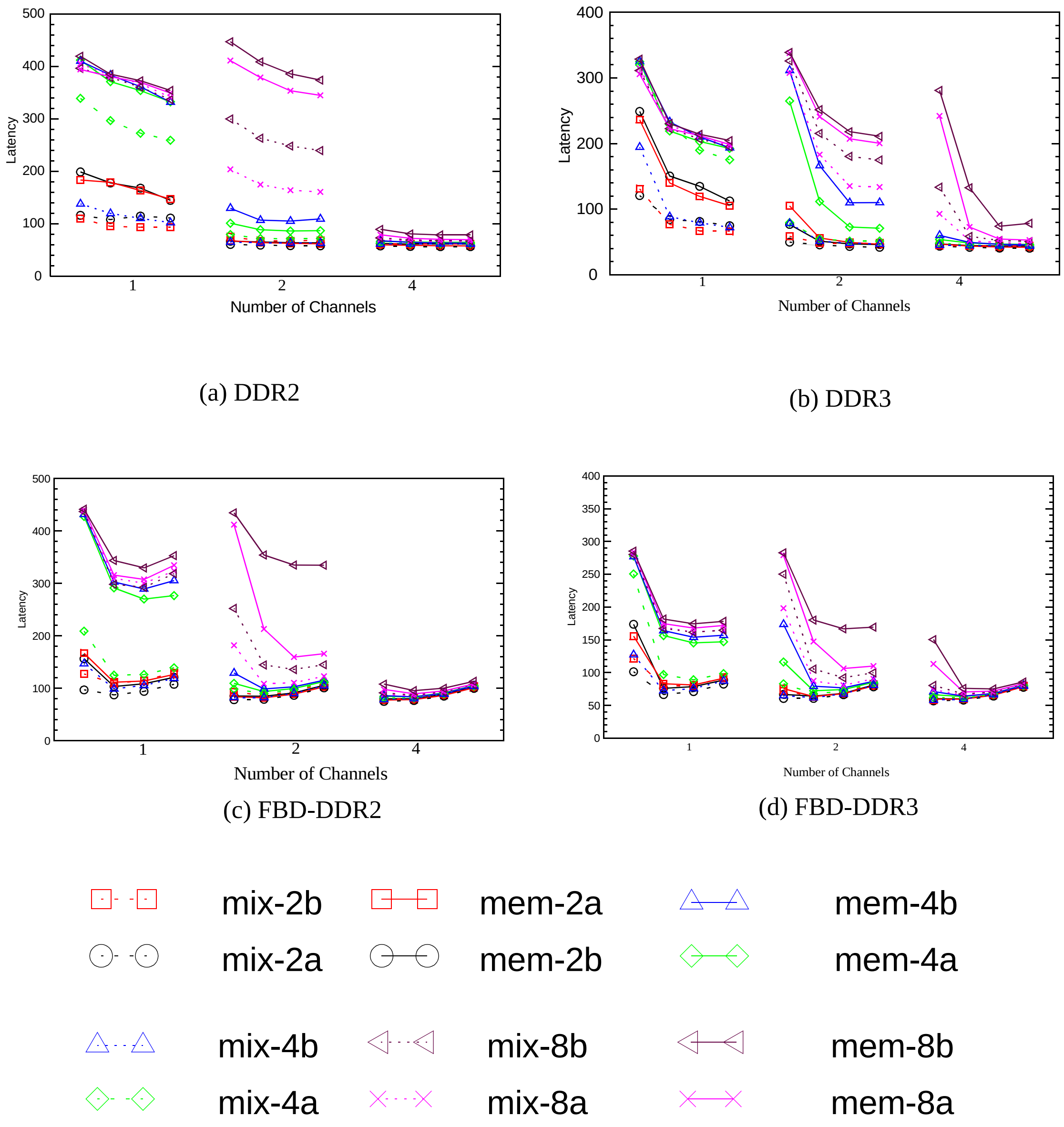
(a) DDR2

(b) DDR3

(c) FBD-DDR2

(d) FBD-DDR3

Figure 4.3. Read Latency Characteristics in an Open Page System. The graphs show the variation in average read latency for a given configuration for the different dram types studied. All data is for a system which uses the open page row buffer management policy, does not use posted CAS and uses a burst length of 8. Each graph plots the average read latency in nano-seconds on the y-axis and the memory system configuration on the x-axis. On the x-axis the system configurations are grouped by the number of channels. Within each channel group, the number of ranks increases as you go from left to right.Note that the y-axis scales for the graphs are different.



(a) DDR2

(b) DDR3

(c) FBD-DDR2

(d) FBD-DDR3

Figure 4.2. Read Latency Characteristics in a Closed Page System. The graphs show the variation in average read latency for the different dram types studied. All data is for a system which uses the closed page row buffer management policy, does not use posted CAS and uses a burst length of 8. Each graph plots the average read latency in nano-seconds on the y-axis and the memory system configuration on the x-axis. On the x-axis the system configurations are grouped by the number of channels. Within each channel group, the number of ranks increases as you go from left to right. Note that the y-axis scales for the graphs are different.

respectively. Note that the average read latency shown in the figures is the average of read latency values observed for all runs for a given benchmark in a particular configuration. In the graphs, systems with identical number of channels are grouped together on the x-axis. Within a group, the points on the left represent topologies with fewer ranks and as one moves from left to right, the number of ranks increases.

Irrespective of dram type, for all the workloads we found that the average read latency typically decreases as more ranks are added in the channel, largely due to the reduction in DRAM level resource conflicts (Figure 4.2 and 4.3). However, FBDIMM systems may see some increase in read latency when going from a 4 deep to an 8 deep channel due to the additional costs of a multi-hop network (Figures 4.2 (c, d) and 4.3 (c, d)). For DDRx systems the increase in ranks essentially comes for free because the DDRx protocols do not provide for arbitrary-length turnaround times.

Latency trends for applications in an open page system (figure 4.3) exhibit greater degrees of non-linearity due to the non-linear variation in the number of DRAM page hits as the number of DRAM resources are increased. Open page FBDIMM systems incur a higher latency cost in comparison to their corresponding DDRx system, then a closed page FBDIMM system and its corresponding DDRx system. This is mainly because due to the reduced cost of a DRAM operation in an open page system, the additional costs of an FBDIMM system are more significant.

The FBDIMM channel by design is associated with higher latency costs due to serialization and the daisy-chaining of DIMMs. For some workloads, as the number of ranks in the channel is increased, FBDIMM systems are able to offset these additional costs by being able to simultaneously schedule transactions to different DIMMs and transmit data to

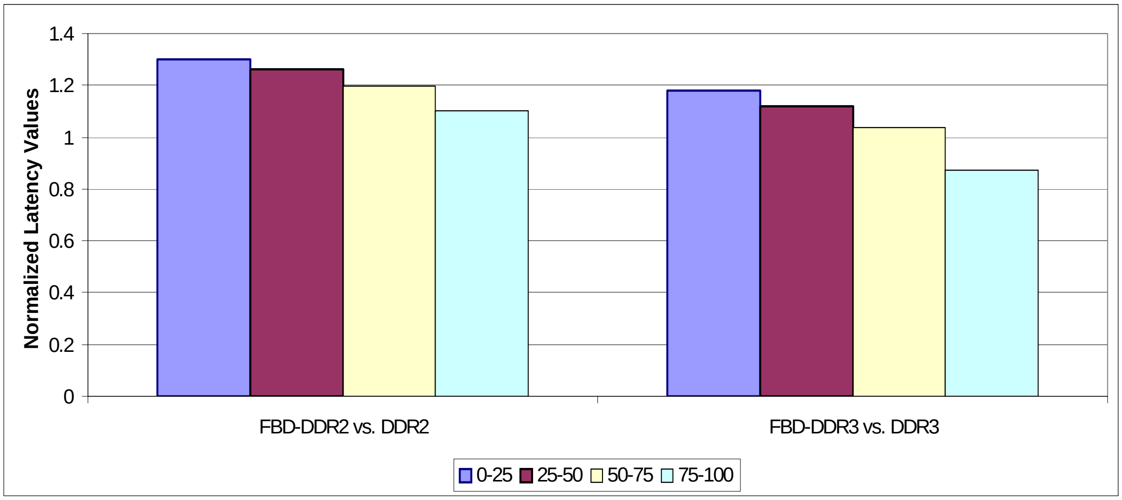


Figure 4.4. FBD vs. DDRx: Latency Improvements as a function of bandwidth utilization. The figure shows the average read latency of an FBD system normalized against the average read latency of its corresponding DDRx system. The latency values are normalized as a function of workload bandwidth utilization. The bandwidth utilization classification is done using the observed bandwidth for the corresponding DDRx system. The normalized average latency is across all workloads and all configurations run.

and from the memory controller. Further, moving the parallel bus off the motherboard and onto the DIMM helps hide the overhead of bus turn-around time. In the case of more lightly loaded systems like some of the multi-channel configurations, due to the lower number of in-flight transactions in a given channel, we find that FBDIMMs are unable to hide the increased transmission costs.

Although, overall FBDIMM systems have 15 to 30% higher latency than its corresponding DDRx system, the relative performance of a FBDIMM system with respect to its DDRx counterpart is a strong function on the system utilization (figure 4.4). System utilization or bandwidth utilization is defined using the equation given below.

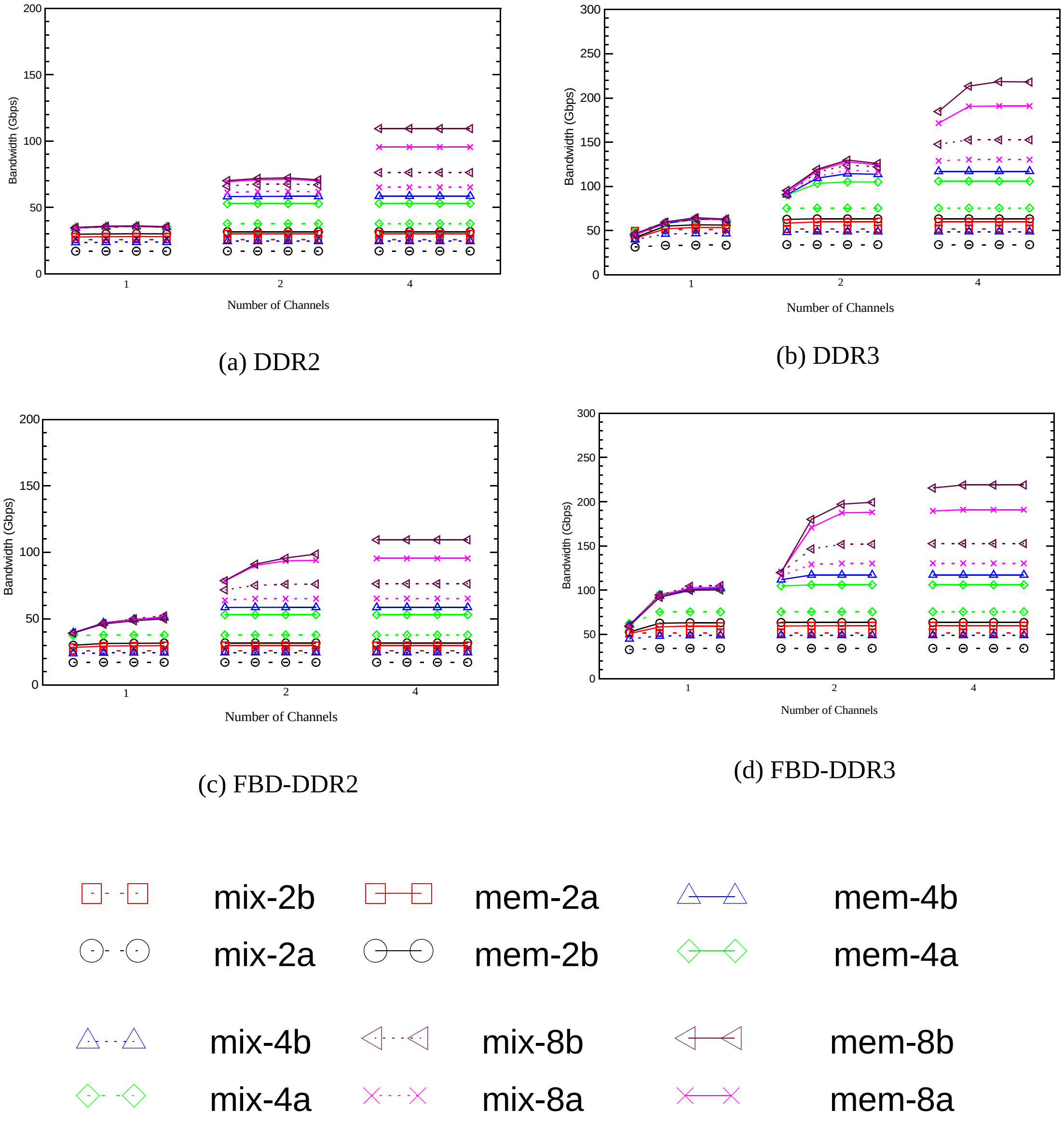
$$BandwidthUtilization = \frac{\text{Observed Bandwidth}}{\text{Total Available Bandwidth}}$$

For the purposes of classification based on bandwidth utilization, we used the observed bandwidth values for the workload on a given system configuration on the DDRx system. Each application was placed into a different utilization bin depending on the range within which its bandwidth utilization fell.

As seen in figure 4.4, at low system utilizations the additional serialization cost of an FBDIMM system is exposed, leading to a relatively higher latency values than in a DDRx system. As system utilization is increased, we find that the difference between the latency of a FBDIMM system and its corresponding DDRx system drops rapidly. In the case of DDR3 based systems, at bus utilizations greater than 75%, the FBDIMM system has lower latency than the conventional system. This is because an FBDIMM system can sustain higher bandwidth utilizations than the DDRx system by effectively utilizing the split-bus architecture. Further, a FBDIMM memory controller is able to exploit the DIMM level parallelism more effectively than a DDRx memory controller because of the separation of the each individual DIMM’s DRAM interface from the common interface it shares with the other DIMMs to communicate with the memory controller. These factors at higher system utilizations lead to better latency characteristics.

Unlike the read latency, we found that the total sustained bandwidth did not vary significantly with changes in the number of ranks in the system (figures 4.5 and 4.6). Sustained bandwidth increases were more pronounced when additional channels were added for workload combinations which could take advantage of the additional available bandwidth e.g. multi-programming workloads with more than 4 programs.

Again as in the case of read latency, we find that the improvement in system bandwidth achieved by using an FBDIMM system is a function of the workloads bandwidth uti-



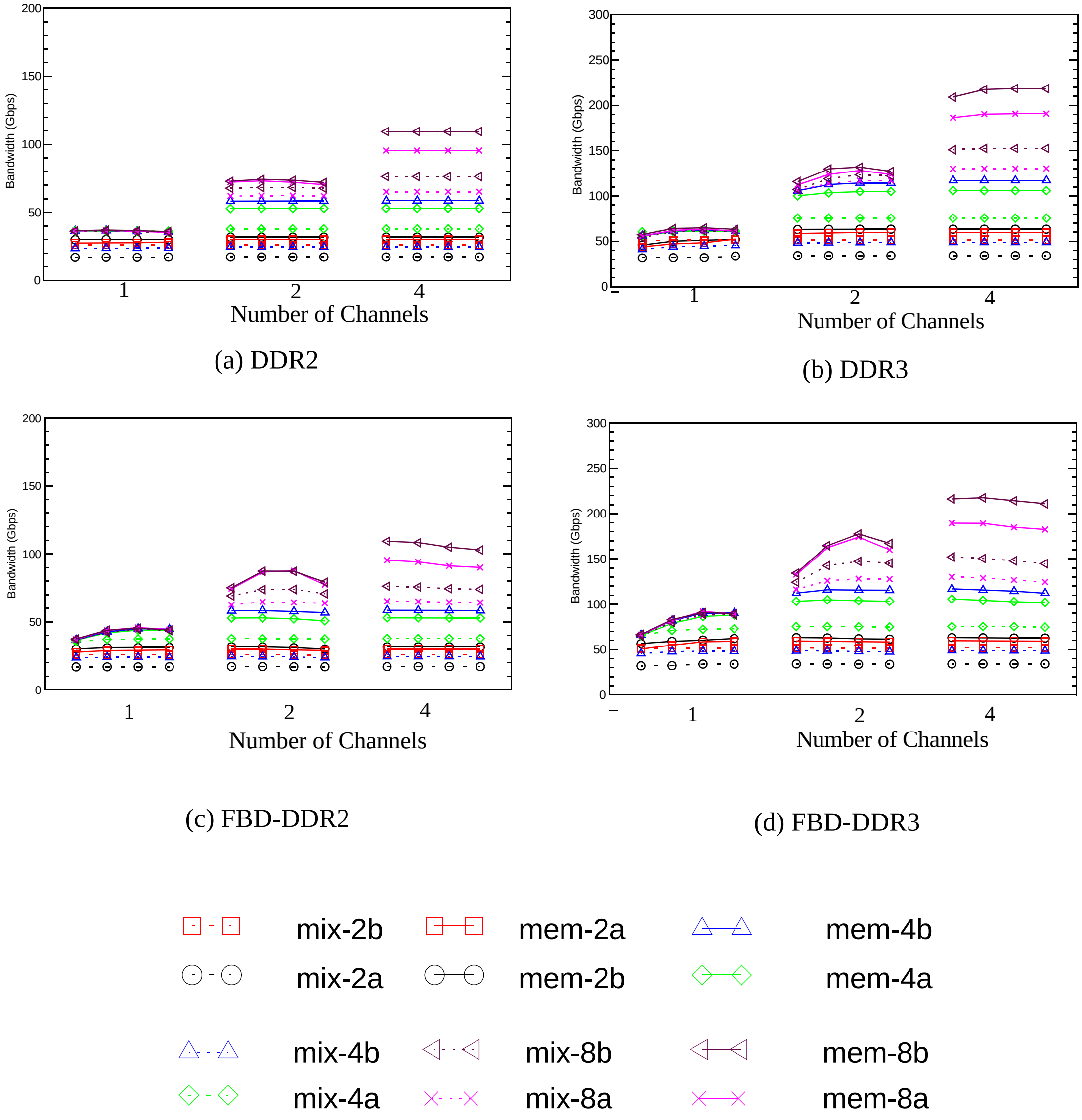
(a) DDR2

(b) DDR3

(c) FBD-DDR2

(d) FBD-DDR3

Figure 4.5. Sustained Bandwidth Characteristics in a Closed Page System. The graphs show the variation in average sustained bandwidth for a given configuration for the different dram types studied. All data is for a system which uses the closed page row buffer management policy, does not use posted CAS and uses a burst length of 8. Each graph plots the average bandwidth in Gbps on the y-axis and the memory system configuration on the x-axis. On the x-axis the system configurations are grouped by the number of channels. Within each channel group, the number of ranks increases as you go from left to right. Note that the y-axis scales for the graphs are different.



(a) DDR2

(b) DDR3

(c) FBD-DDR2

(d) FBD-DDR3

Figure 4.6. Sustained Bandwidth Characteristics in an Open Page System. The graphs show the variation in average sustained bandwidth for a given configuration for the different dram types studied. All data is for a system which uses the open page row buffer management policy, does not use posted CAS and uses a burst length of 8. Each graph plots the average bandwidth in Gbps on the y-axis and the memory system configuration on the x-axis. On the x-axis the system configurations are grouped by the number of channels. Within each channel group, the number of ranks increases as you go from left to right. Note that the y-axis scales for the graphs are different.

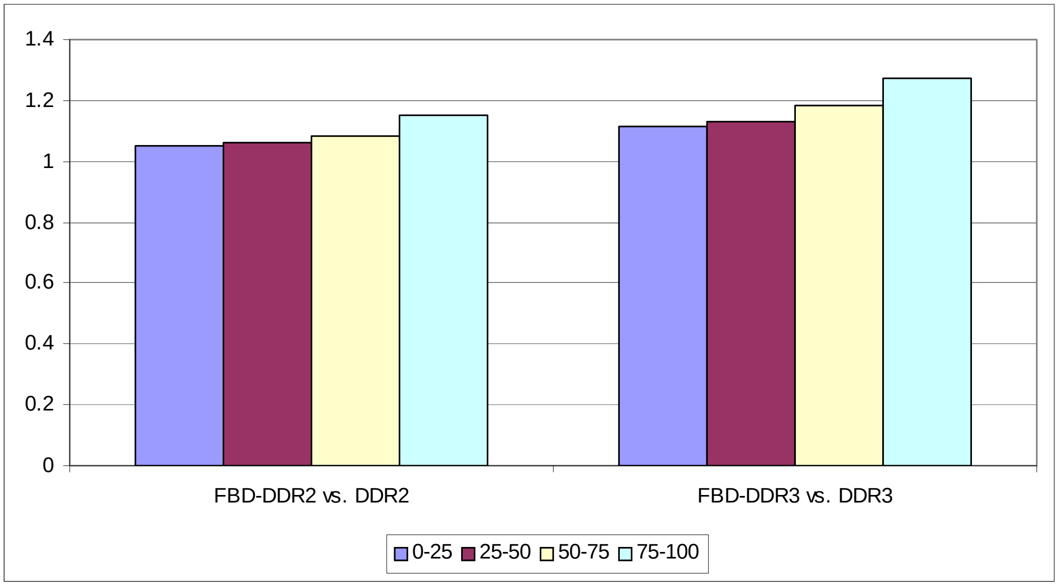


Figure 4.7. FBD vs. DDRx: Bandwidth Improvements as a function of Bandwidth Utilization. The figure shows the overall percentage improvement in bandwidth achieved by a FBD system over its corresponding DDRx system as a function of workload bandwidth utilization. The bandwidth utilization classification is done using the observed bandwidth for the corresponding DDRx system. Note that the latency improvements are calculated for all workloads studied and all configurations run.

lization (Fig 4.7). We found that unlike the latency characteristics, at lower system utilizations i.e. less than 25% for DDR2 based systems and 50% for DDR3 based systems, that the bandwidth seen in a FBDIMM system is nearly similar that seen in its corresponding DDRx system. As system utilization demand increases, we found that by utilizing the additional system bandwidth available in an FBDIMM system, the workloads experience better bandwidth characteristics on a FBDIMM system than on its corresponding DDRx based system. Busier workloads tend to use both FBD channels simultaneously, whereas the reads and writes have to take turns in a DDRx system. Bandwidth utilization drops as the number of channels in the system are increased. Consequently as the number of channels increased, the gains in bandwidth diminish till they vanish completely.

4.2.2 Latency Contributors

In this section, we look at the trends in latency, and the factors impacting them, in further detail. The overall behavior of the contributors to the read latency are characteristic of the system topology and are not particular to a given scheduling policy or row buffer management policy. Hence, we use a particular row buffer management policy, RIFF in the case of closed page systems and OBF in case of open page systems for the discussion.

(a) MIX-2b-DDR2-CP

(b) MEM-2b-DDR2-CP

Figures 4.8 to 4.15, shows the average read latency divided into queueing delay overhead and the transaction processing overhead. The queueing delay component refers to the duration the transaction waited in the queue for one or more resources to become available. For FBDIMM systems, the causes for queueing delay include memory controller request queue availability, south link availability, DIMM availability (including on-DIMM command and data buses and bank conflicts), and north link availability. In the case of DDRx systems, the causes for queueing delay include memory controller request queue availability, command bus availability, data bus availability and DIMM availability (in this case, mainly bank availability). Note that the overlap of queueing delay is monitored for all components except the memory controller request queue factor. The default latency cost is the cost associated with making a read request in an unloaded channel.

(c) MIX-4b-DDR2-CP

(d) MEM-4b-DDR2-CP

The queueing delay experienced by a read transaction is rarely due to any single factor, but usually due to a combination of factors. Changing the system configuration, be it by adding more ranks in the channel or increasing the number of channels in the system, result in a change in the availability of all the resources to a different degree. Thus, we see that the latency trends due to changes in a configuration are affected by how exactly these individual queueing delays change.

(e) MIX-8b-DDR2-CP

(f) MEM-8b-DDR2-CP

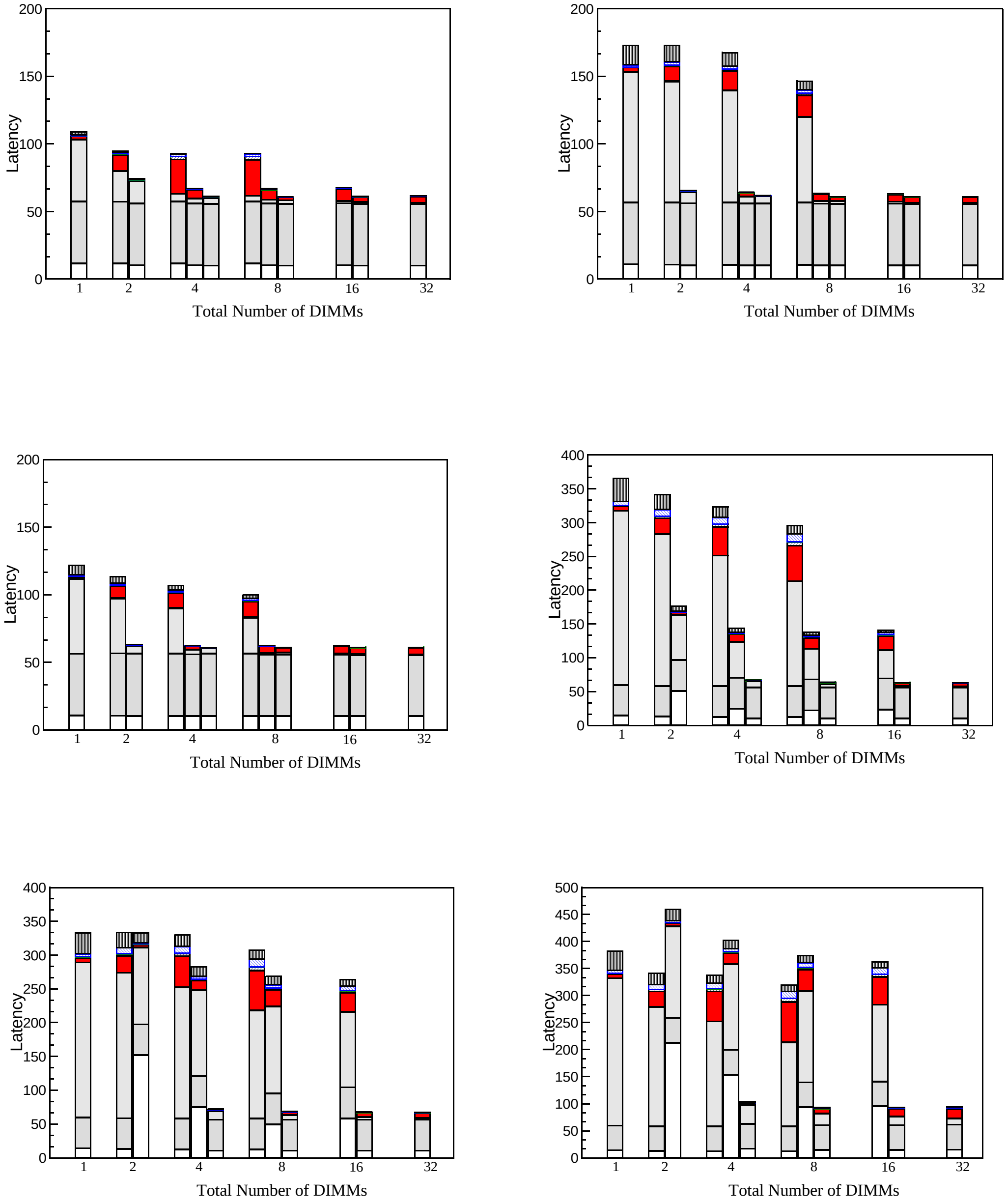
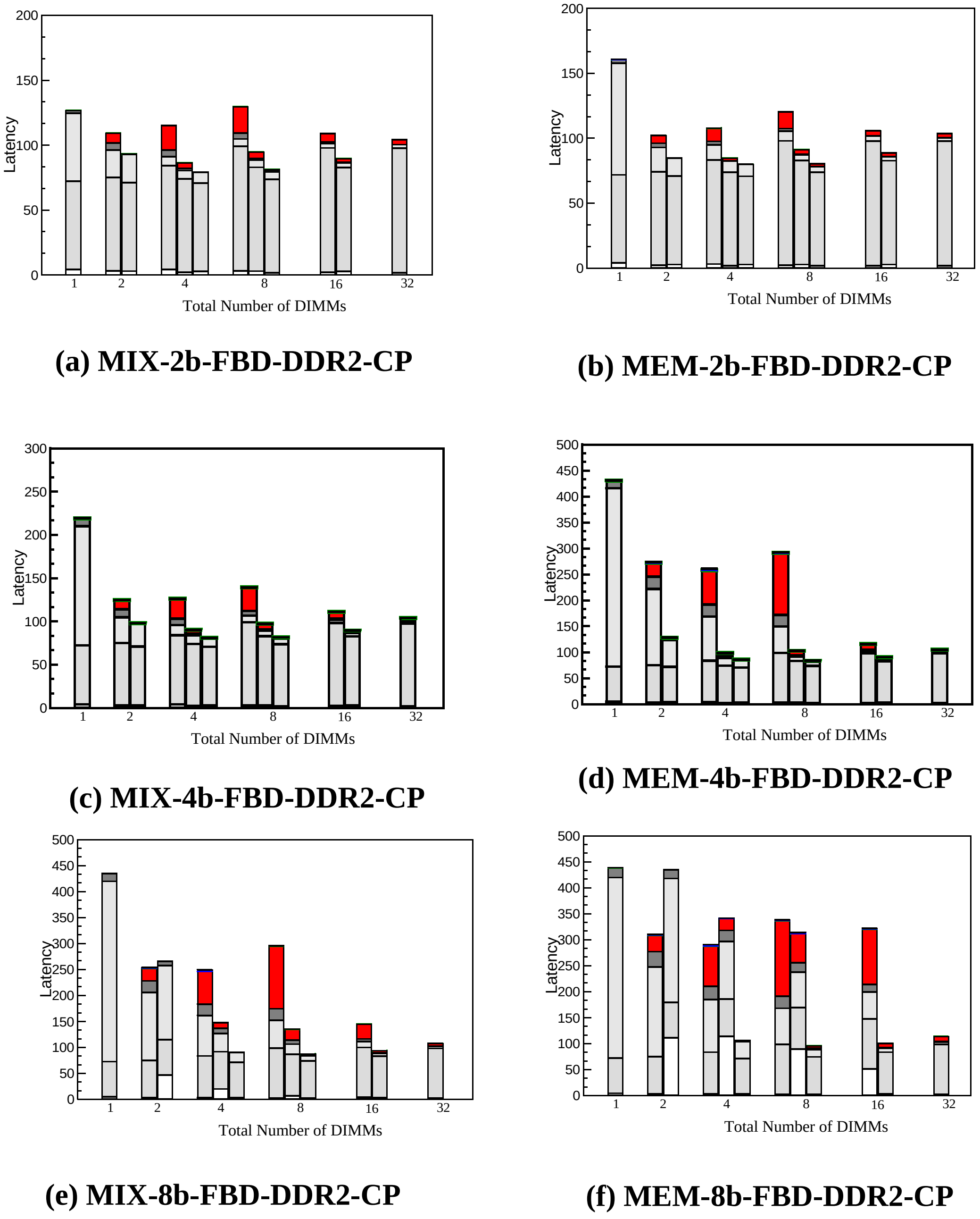


Figure 4.8. Latency Contributors for DDR2 Closed Page Systems. Systems with identical numbers of dimms are grouped together. Within a group, the bars on the left represent topologies with fewer numbers of channels. The dram configuration was FBD-DDR2 using fixed latency mode. The y-axis shows latency in nano-



(a) MIX-2b-FBD-DDR2-CP

(b) MEM-2b-FBD-DDR2-CP

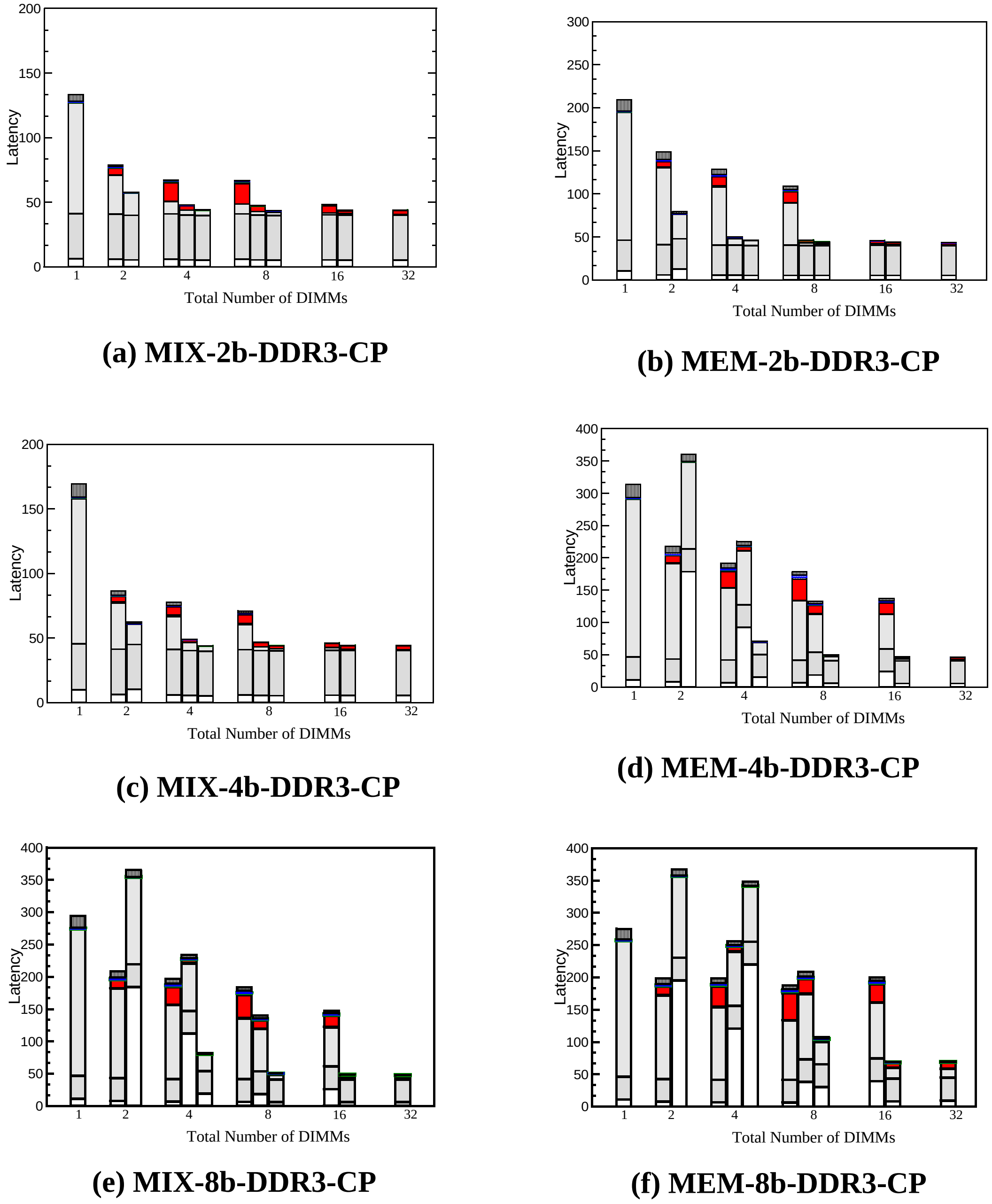
(c) MIX-4b-FBD-DDR2-CP

(d) MEM-4b-FBD-DDR2-CP

(e) MIX-8b-FBD-DDR2-CP

(f) MEM-8b-FBD-DDR2-CP

Figure 4.9. Latency Contributors for FBD-DDR2 Closed Page Systems. Systems with identical numbers of dimms are grouped together. Within a group, the bars on the left represent topologies with fewer numbers of channels. The dram configuration was FBD-DDR2 using fixed latency mode. The y-axis shows latency in nano-seconds. The y-axis scales are different for each graph.



(a) MIX-2b-DDR3-CP

(b) MEM-2b-DDR3-CP

(c) MIX-4b-DDR3-CP

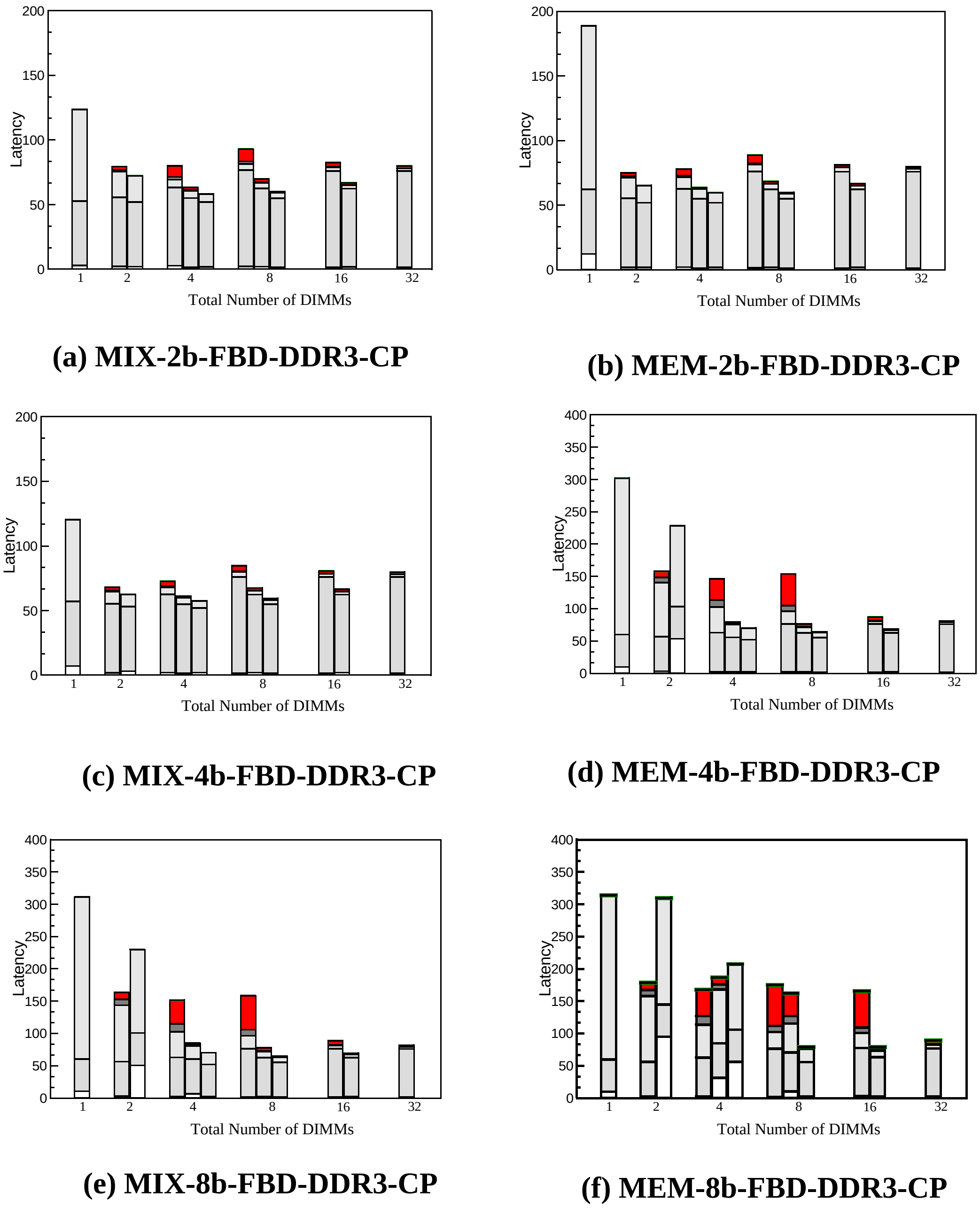
(d) MEM-4b-DDR3-CP

(e) MIX-8b-DDR3-CP

(f) MEM-8b-DDR3-CP

Figure 4.10. Latency Contributors for DDR3 Closed Page Systems. Systems with identical numbers of dimms are grouped together. Within a group, the bars on the left represent topologies with fewer numbers of channels. The dram configuration was FBD-DDR2 using fixed latency mode. The y-axis shows latency in nano-seconds. The y-axis scales are different for each graph.re calculated for all workloads studied and all configurations run.

- ▦ Command Bus and DIMM
- ▨ Command Bus
- 🟩 Command Bus, DIMM, Data Bus
- ▨ Command Bus, Data Bus
- 🟥 Data Bus
- ■ DIMM, Data Bus
- □ DIMM
- □ Default Latency
- □ Transaction Queue



(a) MIX-2b-FBD-DDR3-CP

(b) MEM-2b-FBD-DDR3-CP

(c) MIX-4b-FBD-DDR3-CP

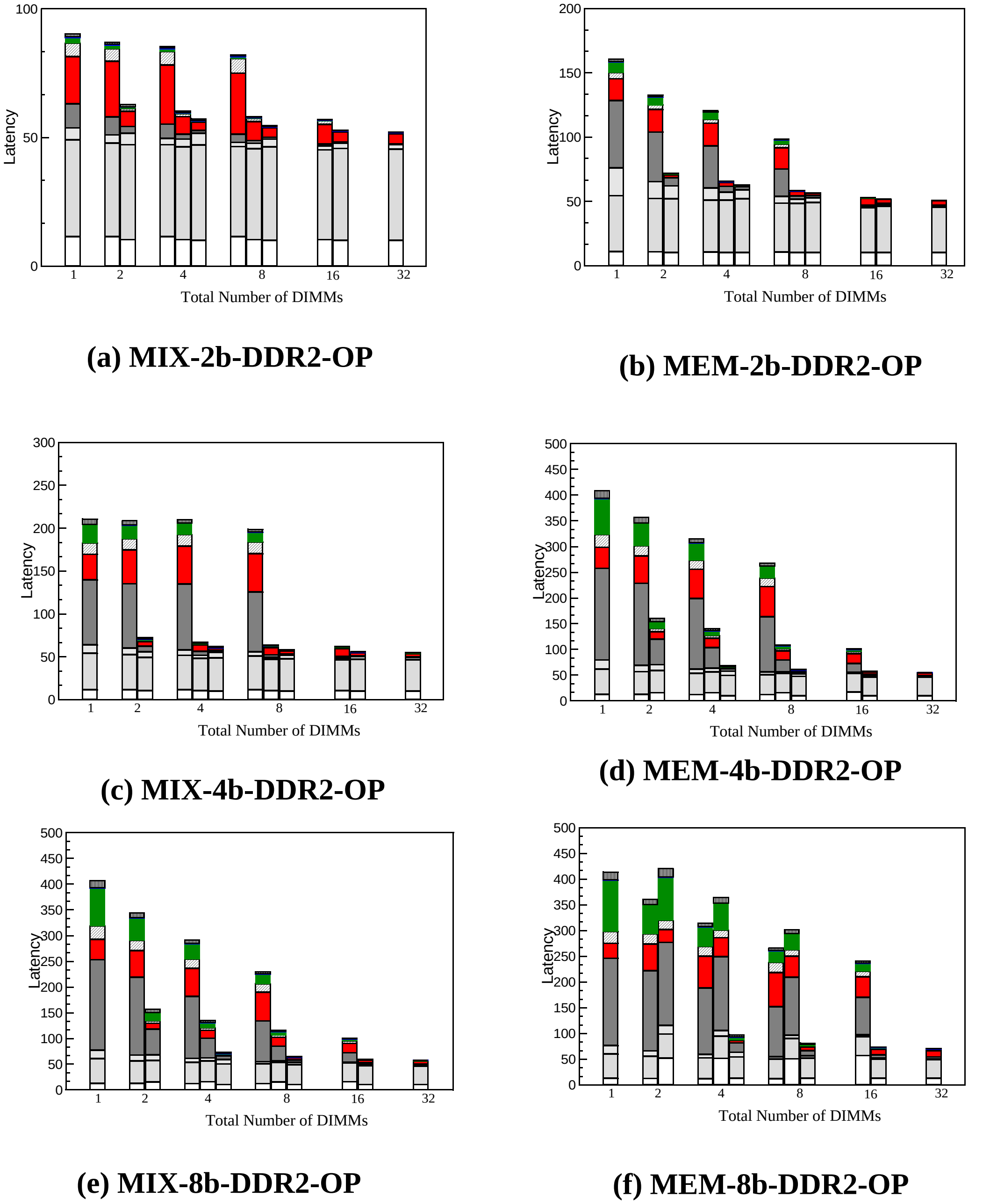
(d) MEM-4b-FBD-DDR3-CP

(e) MIX-8b-FBD-DDR3-CP

(f) MEM-8b-FBD-DDR3-CP

Figure 4.11. Latency Contributors for FBD-DDR3 Closed Page Systems. Systems with identical numbers of dimms are grouped together. Within a group, the bars on the left represent topologies with fewer numbers of channels. The dram configuration was FBD-DDR3 using fixed latency mode. The y-axis shows latency in nanoseconds. The y-axis scales are different for each graph.

▩ South Link and DIMM
▨ South Link
🟩 South Link, DIMM, North Link
▨ South Link, North Link
🟥 North Link
■ DIMM, North Link
□ DIMM
□ Default Latency
□ Transaction Queue



(a) MIX-2b-DDR2-OP

(b) MEM-2b-DDR2-OP

(c) MIX-4b-DDR2-OP

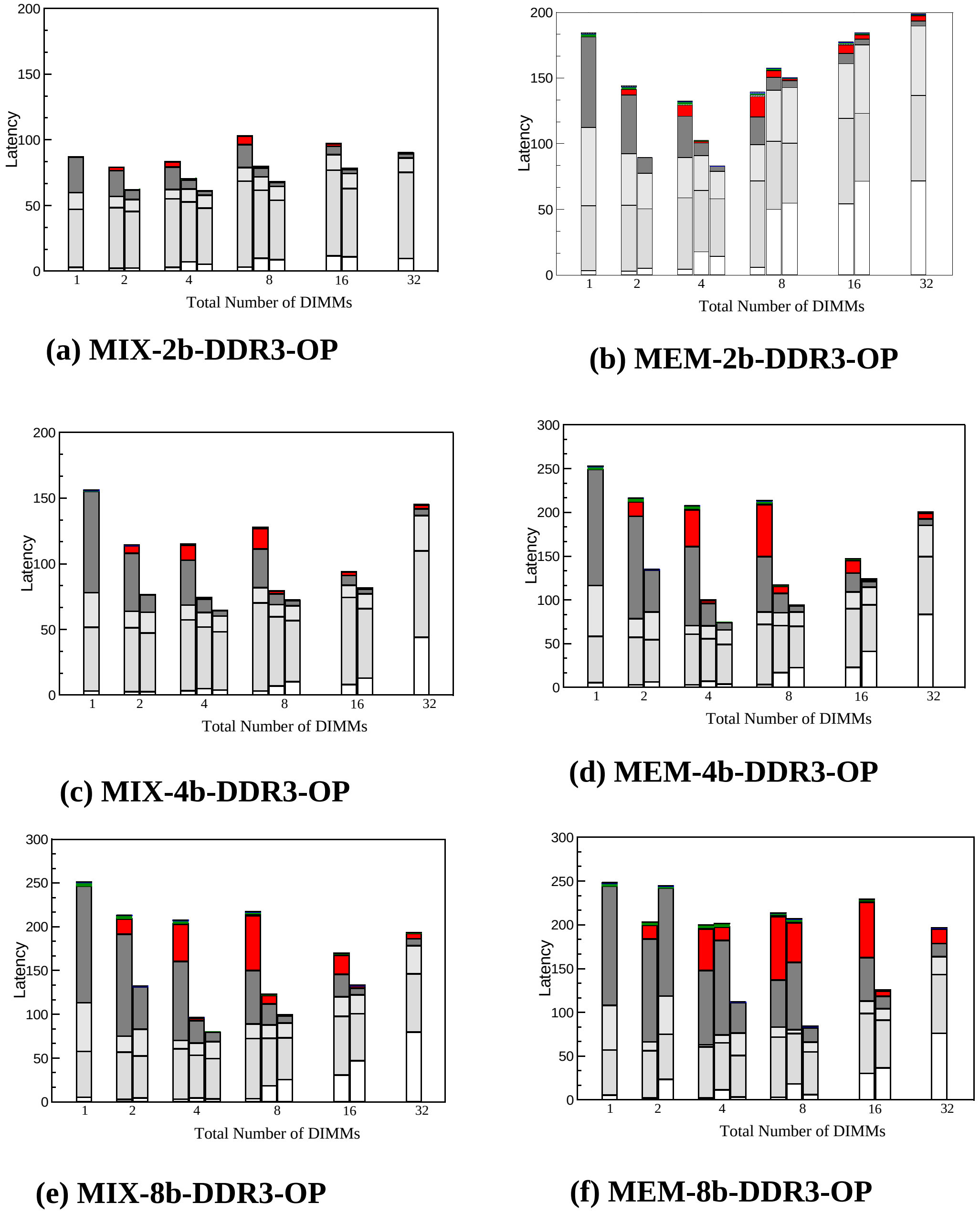
(d) MEM-4b-DDR2-OP

(e) MIX-8b-DDR2-OP

(f) MEM-8b-DDR2-OP

Figure 4.12. Latency Contributors for DDR2 open page systems. Systems with identical numbers of dimms are grouped together. Within a group, the bars on the left represent topologies with fewer numbers of channels. The y-axis shows latency in nano-seconds. The y-axis scales are different for each graph.

▦ Command Bus and DIMM
▨ Command Bus
🟩 Command Bus, DIMM, Data Bus
▨ Command Bus, Data Bus
🟥 Data Bus
■ DIMM, Data Bus
□ DIMM
□ Default Latency
□ Transaction Queue



(a) MIX-2b-DDR3-OP

(b) MEM-2b-DDR3-OP

(c) MIX-4b-DDR3-OP

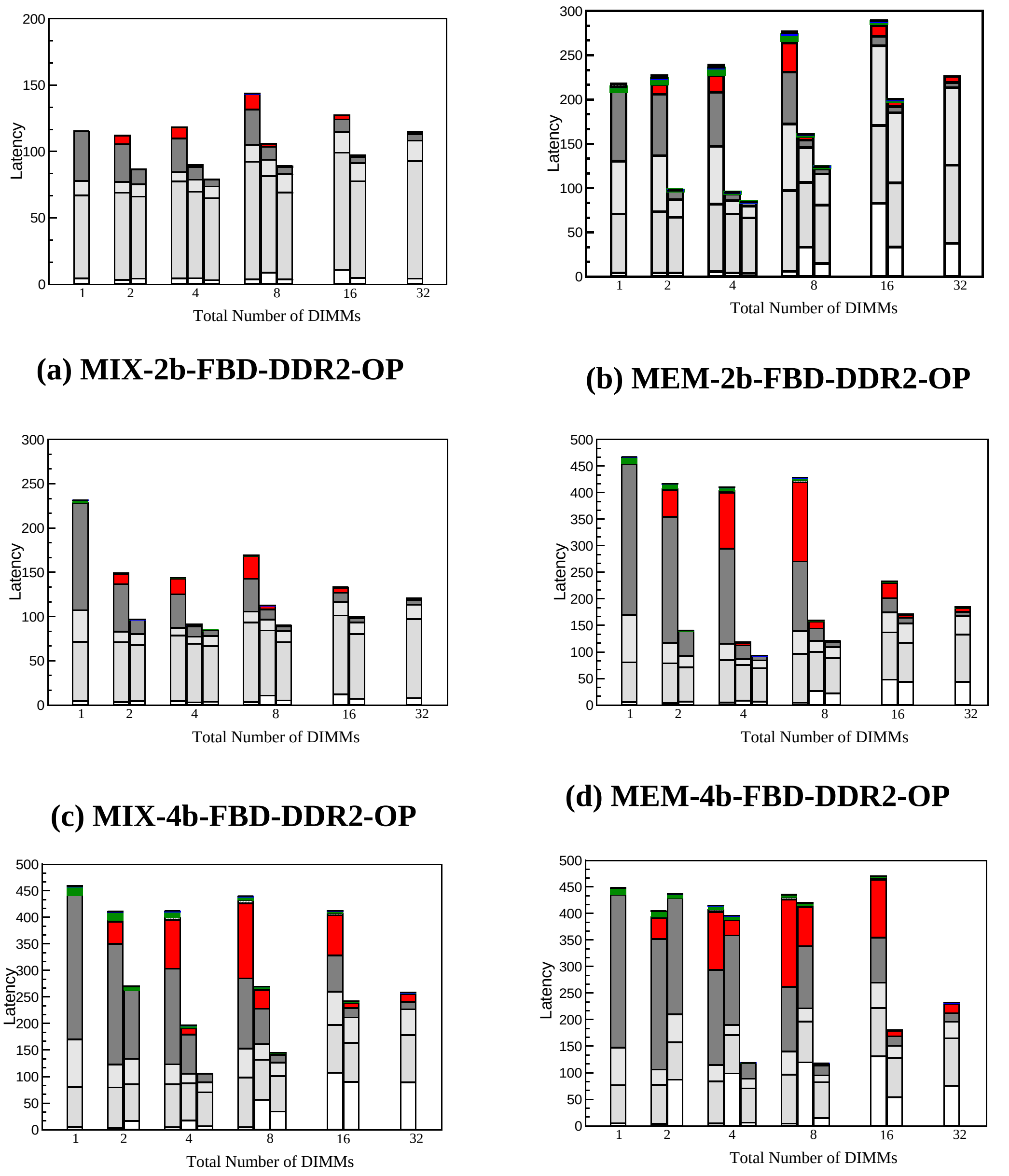
(d) MEM-4b-DDR3-OP

(e) MIX-8b-DDR3-OP

(f) MEM-8b-DDR3-OP

Figure 4.14. Latency Contributors for DDR3 open page systems. Systems with identical numbers of dimms are grouped together. Within a group, the bars on the left represent topologies with fewer numbers of channels. The y-axis shows latency in nano-seconds. The y-axis scales are different for each graph.

▨ Command Bus and DIMM
▨ Command Bus
🟩 Command Bus, DIMM, Data Bus
▨ Command Bus, Data Bus
🟥 Data Bus
■ DIMM, Data Bus
□ DIMM
□ Default Latency
□ Transaction Queue



(a) MIX-2b-FBD-DDR2-OP

(b) MEM-2b-FBD-DDR2-OP

(c) MIX-4b-FBD-DDR2-OP

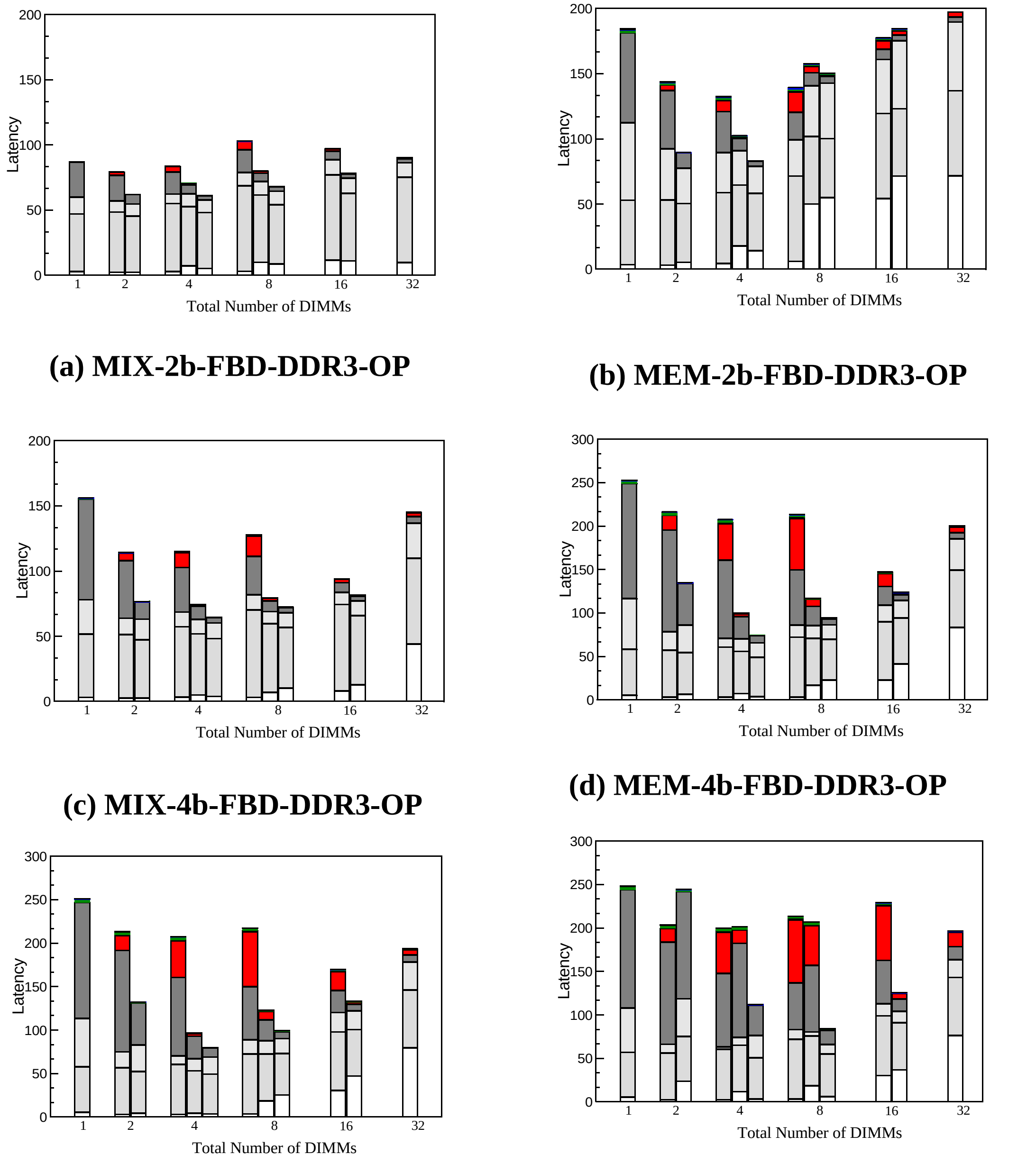
(d) MEM-4b-FBD-DDR2-OP

(e) MIX-8b-FBD-DDR2-OP

(f) MEM-8b-FBD-DDR2-OP

Figure 4.13. Latency Contributors for FBD-DDR2 open page systems. =Systems with identical numbers of dimms are grouped together. Within a group, the bars on the left represent topologies with fewer numbers of channels. The dram configuration was FBD-DDR2 using fixed latency mode. The y-axis shows latency in nano-seconds. The y-axis scales are different for each graph.

▦ South Link and DIMM
▨ South Link
🟩 South Link, DIMM, North Link
▨ South Link, North Link
🟥 North Link
■ DIMM, North Link
□ DIMM
□ Default Latency
□ Transaction Queue



(a) MIX-2b-FBD-DDR3-OP

(b) MEM-2b-FBD-DDR3-OP

(c) MIX-4b-FBD-DDR3-OP

(d) MEM-4b-FBD-DDR3-OP

(e) MIX-8b-FBD-DDR3-OP

(f) MEM-8b-FBD-DDR3-OP

Figure 4.15. Latency Contributors for FBD-DDR3 open page systems. Systems with identical numbers of dimms are grouped together. Within a group, the bars on the left represent topologies with fewer numbers of channels. The dram configuration was FBD-DDR3 using fixed latency mode. The y-axis shows latency in nano-seconds. The y-axis scales are different for each graph.

Typically, single-rank configurations have higher latencies due to insufficient number of memory banks to distribute requests to. In such systems the DIMM is the dominant system bottleneck and can contribute as much as 50% of the overall transaction queueing delay. Adding more ranks in these system can help reduce the DIMM-based queueing delays. The reductions are 20-60%, when going from a one rank to a two-rank channel. For 4-8 rank channel, the DIMM-based queueing delay is typically only 10% of that in the single-rank channel. However, in a DDRx system the latter is not always true and we see that in Fig 4.8 (b, d, e, f) 4.10(d, e, f), for a single channel case that the DIMM related overheads do not reduce significantly as the channel is made longer. This is because although there are multiple DIMMs available, the memory controller is not able to take advantage of the DIMM level parallelism like a FBDIMM controller can. In an FBDIMM system, the DRAM interface is isolated from the interface between the memory controller and the DIMMs. Further the use of a split data bus architecture and the ability of the memory controller to schedule multiple commands simultaneously in the same DRAM command clock cycle, enables the memory controller to effectively exploit the DIMM-level parallelism. Hence we see that although there are significant reductions in DIMM associated queueing delay in an FBDIMM channel, this is not the case of a DDRx system. Increasing the number of ranks while simultaneously increasing the number of channels reduces the DIMM associated queueing delay by enabling the DDRx controller to use the channel level parallelism to take advantage of the DIMM level parallelism.

The interaction of the increase in processing overhead, number of in-flight transactions using the channel, the type of these transactions and the lowering of DIMM-level conflicts result in two prominent latency trends namely either a linear increase in latency with increased channel lengths or a more U-shaped latency trend i.e. an initial drop in latency followed by an increase. This is unlike a DDRx system where the increase in the channel depth usually results in a decrease in latency.

In an FBDIMM system, increasing the channel depth raises the transmission costs for frames on the channel. This results in an increase in the read latency with increasing channel depth for the MIX-2 and MIX-4 workloads at larger channel widths when the utilization is low (Fig 4.9(a, c), Fig 4.11(a, c), Fig 4.13(a, c) and Fig 4.15 (a, c)). Interestingly, in a single channel case, the additional DRAM-level parallelism available in a deeper channel can counter the rise in frame transmission costs sufficiently to reduce overall latency (e.g Fig. 4.9 (d)). The gains in parallelism are especially apparent when going from a single-rank to a two-rank channel. These gains gradually taper off for further increases in channel depth.

With deeper channels, the increased number of in-flight transactions results in an increase in the queueing delay due to the link-unavailability. This is attributable to the heightened competition for the bus due to the additional number of in-flight transactions. This increase can combine with the increase in processing cost to offset the gains due to increased DIMM-level parallelism. Thus, we see that for nearly all the workloads in a single channel configuration, that the latency increases in a single channel system when the channel depth is increased from 4 to 8 (Fig. 4.9 (a-f) and Fig. 4.11 (a-f)).

As expected increasing the number of channels results in the reduction in latency by increasing the amount of channel bandwidth available. We find that for the workloads with higher bandwidth requirements like MEM-8 and MIX-8 workloads, that doubling the number of channels can result in an increase in read latency (Fig 4.9(e, f), Fig. 4.11(e, f). These increases are mainly due to increases in queueing delay due to the transaction queue being unavailable. The memory architecture modelled has a common shared BIU which has as many slots as the combined total of the number of transaction queue slots across all the channels. Consequently, when a particular channel's transaction queue fills up and there are unissued requests waiting in the BIU for that channel, the BIU quickly fills up and these requests experience significant queueing delay waiting for the transaction queue to become available.

The main difference between open page systems and closed page systems (compare Figures 4.9 and 4.13, Figures 4.8 and 4.12, Figures 4.10 and 4.14 and Figures 4.11 and 4.15), is that open page configurations experience higher queueing delay due to the unavailability of multiple resources e.g. DIMM and return data link, as compared to closed page systems which experience higher queueing delay due to the unavailability of mainly a single resource.

4.2.3 Impact of Scheduling Policy

We found that the RIFF scheduling policy had the best latency characteristics overall. In an open page system, the OBF scheduling policy did pretty well too (figure 4.20). With regards to bandwidth utilization, we found that the differences between the performance of the various scheduling policies less obvious. In an open page system, the OBF scheduler had the highest overall bandwidth, while in a closed page system it was the greedy scheduler (figure 4.22). Overall FBDIMM systems and DDR systems had the same scheduling policies perform better, although discrepancies existed for what performed best in a particular configuration.

Figures 4.16, 4.17 and 4.18 provide a pictorial representation of the scheduling policy having the lowest latency for different system topologies for the workloads studied. Note that only scheduling policies which are more than 2.5% of the average across all scheduling policies are shown in the figure. A histogram of how often a particular scheduling policy had the lowest latency possible, irrespective of how much lower, is shown in figure 4.19. From the two different representations, we can see that the RIFF policy, which prioritizes read requests over write requests, performs the best for most of the cases. In an open page system the RIFF scheduling policy also does better than a scheduling policy like OBF, which attempts to reduce read latency by exploiting DRAM locality. The OBF scheduler can delay servicing a read request to handle an earlier issued write command which experiences a page hit. The RIFF scheduler on the other hand prioritizes the read traffic, thereby improving the latency of a read transaction, which is what we focus on.

It was interesting that the RIFF scheduling policy performed better than the other scheduling policies in a FBDIMM system despite the separation of the read and write data paths which reduces the competition between read and write traffic for the channel databus. However, since the command bus and write-data bus are shared in FBDIMM systems, read transactions compete with write transactions for command bandwidth. Read transactions also compete with write transactions that are destined for the same rank or DIMM. Hence, despite the separation of the read and write data paths, these two factors make prioritizing read transactions still an effective policy in FBDIMM systems.

In the case of bandwidth characteristics, we found that additional channels in the system had a more pronounced impact on bandwidth than any other parameter. The OBF and

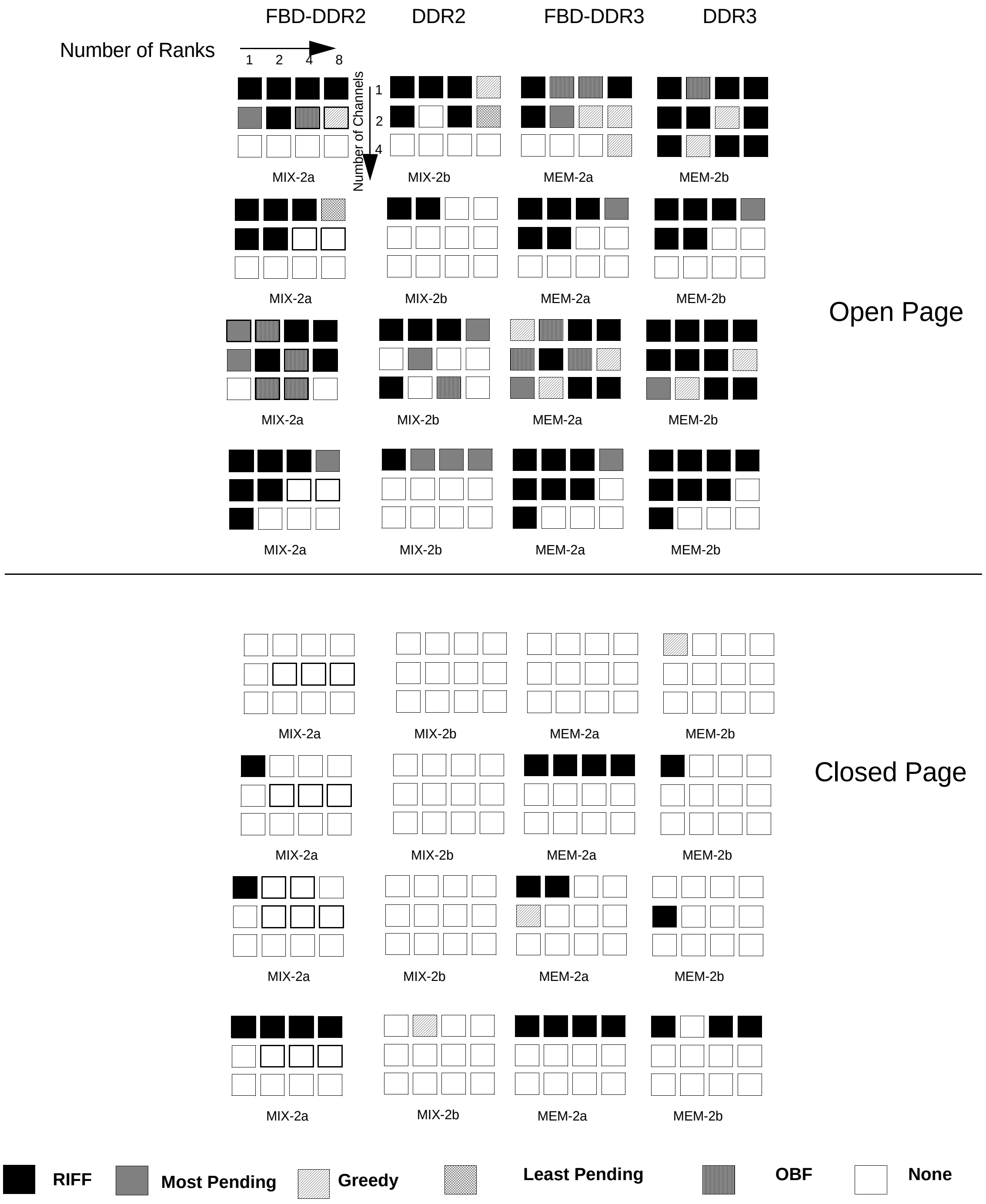


Figure 4.16. Best Scheduling Policy for Average Read Latency. The figure show which scheduling policy had the best latency characteristics for each differer configurations for the various applications studied.A scheduling policy emerges as th winner if the observed mean latency for the given configuration is 2.5% or more bette than the worst performing scheduling policy for a given configuration point. Note tha none implies that no clear winner emerged. All data for FBDIMM is in fixed latenc mode

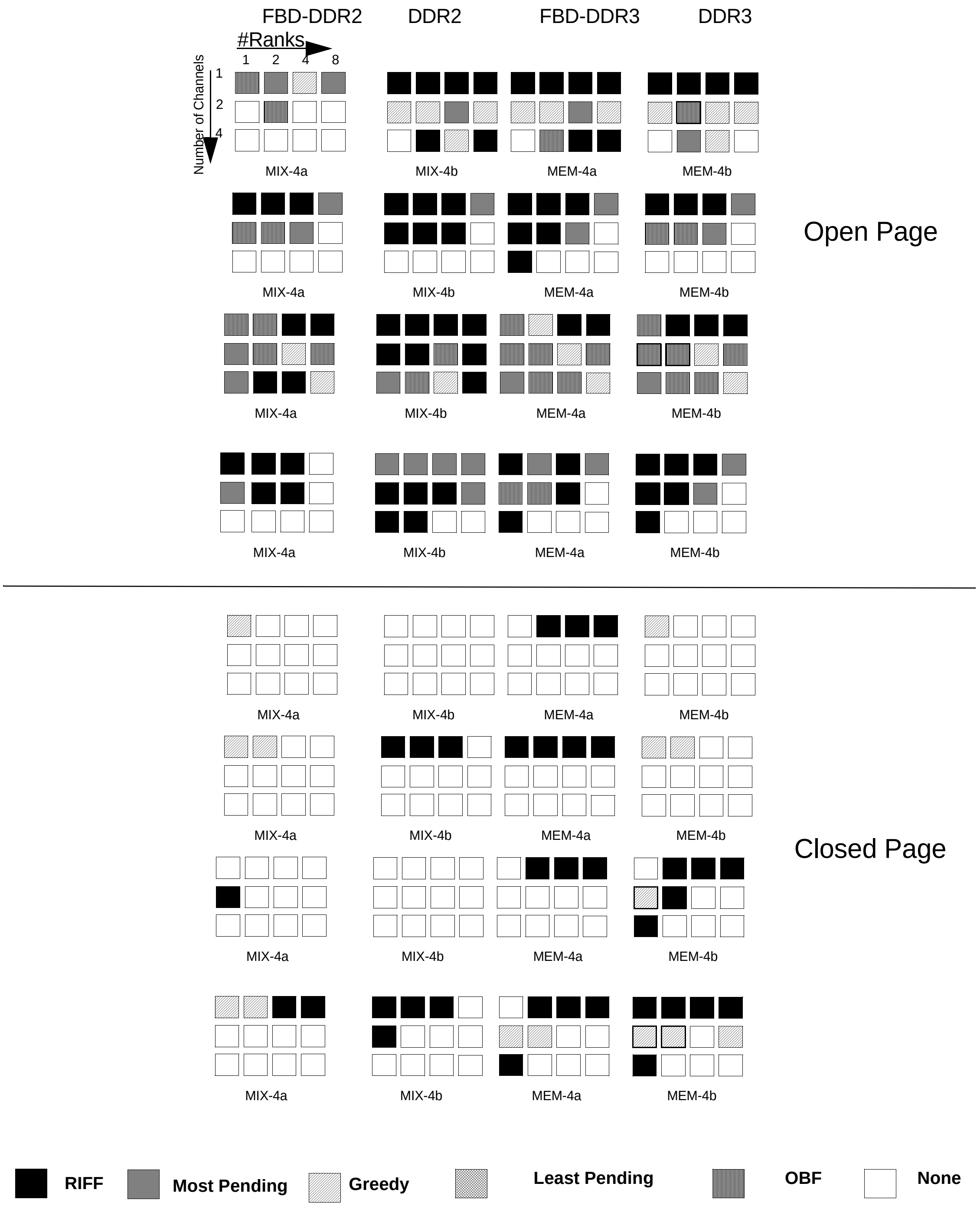


Figure 4.17. Best Scheduling Policy for Average Read Latency. The figure shows which scheduling policy had the best latency characteristics for each different configurations for the various applications studied.A scheduling policy emerges as the winner if the observed mean latency for the given configuration is 2.5% or more better than the worst performing scheduling policy for a given configuration point. Note that none implies that no clear winner emerged. All data for FBDIMM is in fixed latency mode

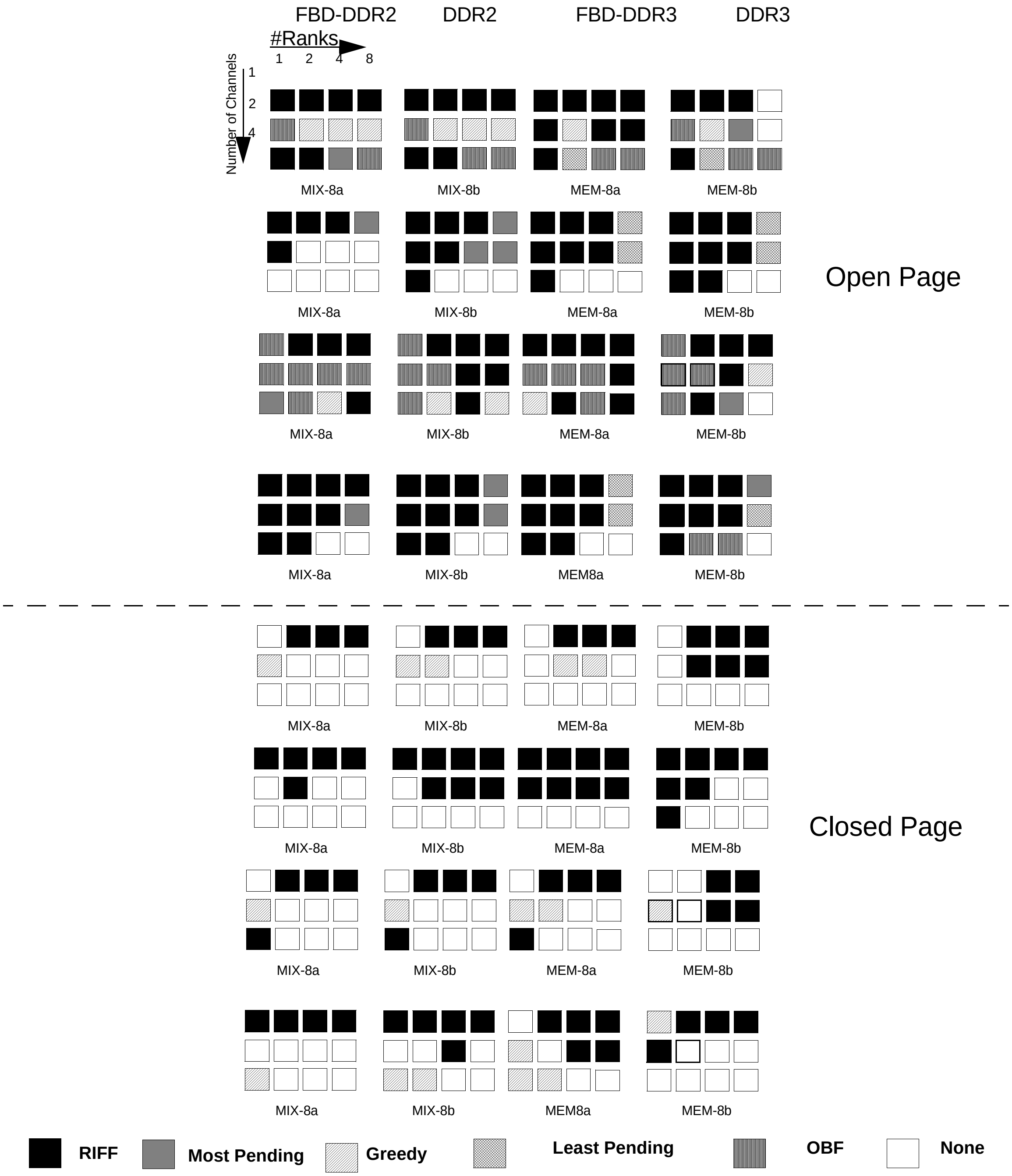
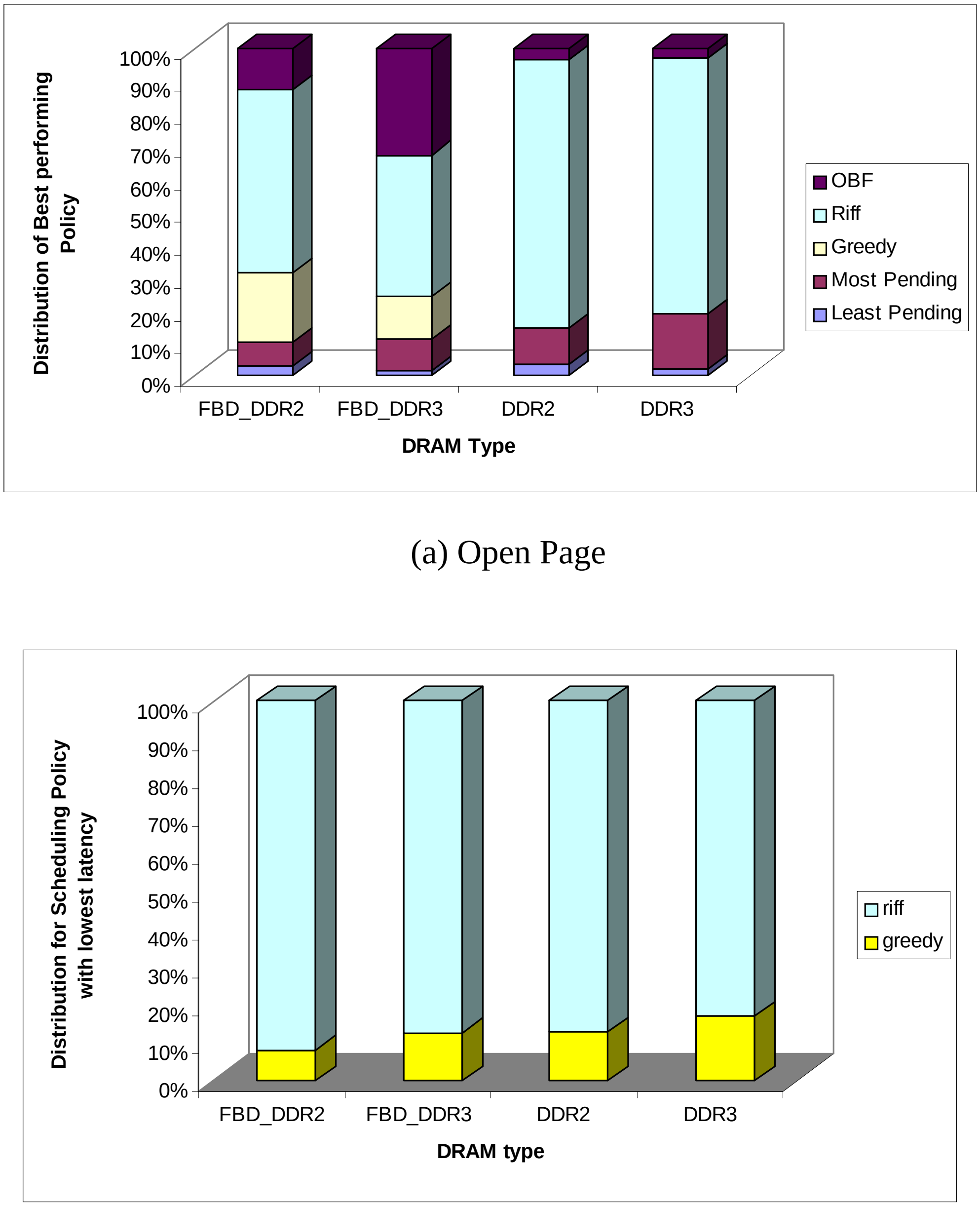


Figure 4.18. Best Scheduling Policy for Average Read Latency. The figure shows which scheduling policy had the best latency characteristics for each different configurations for the various applications studied.A scheduling policy emerges as the winner if the observed mean latency for the given configuration is 2.5% or more better than the worst performing scheduling policy for a given configuration point. Note that none implies that no clear winner emerged. All data for FBDIMM is in fixed latency mode



(a) Open Page

(b) Closed Page

Figure 4.19. Histogram of scheduling policy with lowest read latency. The figure shows the distribution of which scheduling policy had the best latency characteristics across all the configurations studied. The data is shown for each DRAM type (plotted on the x-axis).

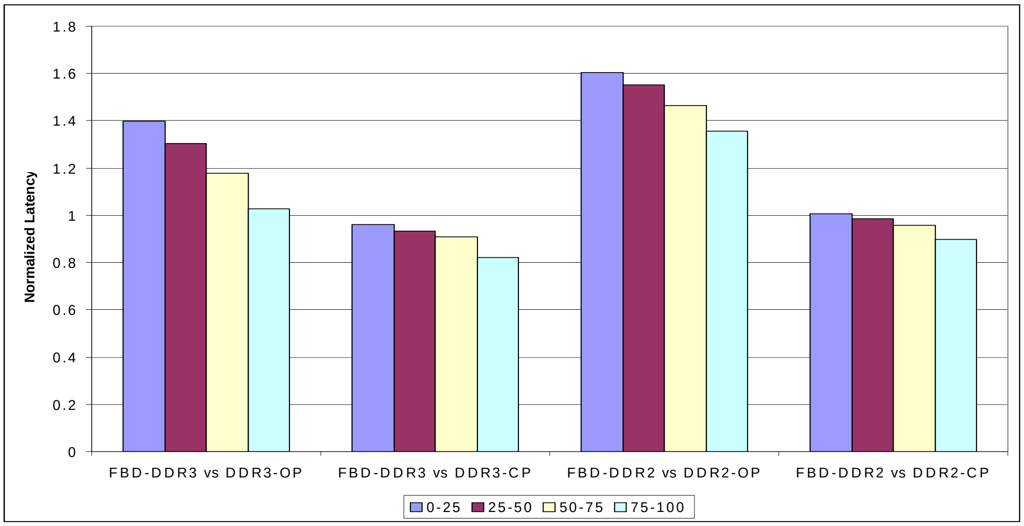
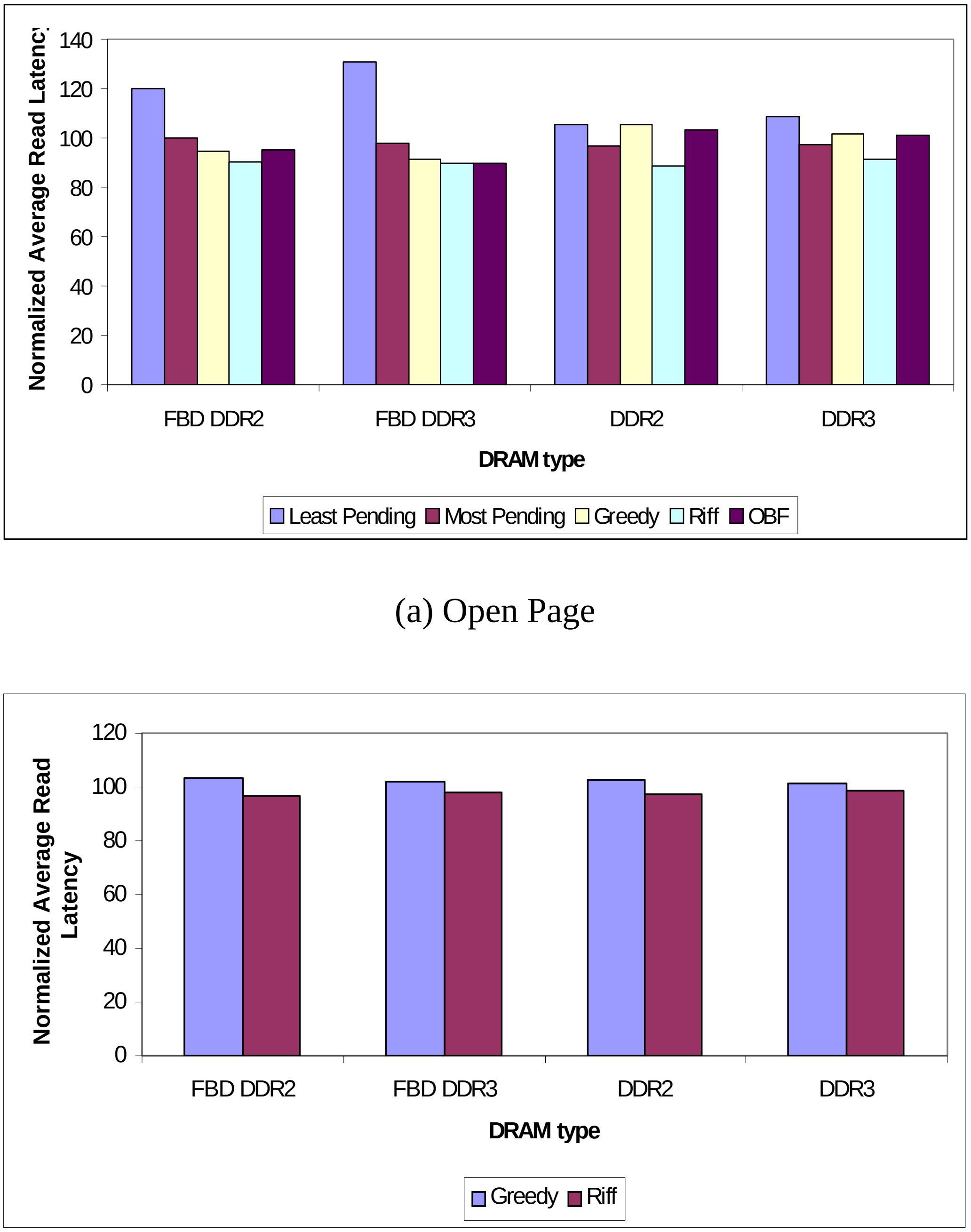


Figure 4.23. FBD vs. DDRx: Latency Improvements for different row buffer management policies. The figure shows the change in read latency achieved by a FBD system over its corresponding DDRx system as a function of workload bandwidth utilization for different row buffer management policies. The bandwidth utilization classification is done using the observed bandwidth for the corresponding DDRx system. Note that the latency improvements are plotted as normalized against the latency of a DDRx system. Latency averages are calculated for all workloads studied and all configurations run.

greedy scheduling policy had better bandwidth characteristics, while the RIFF scheduling policy which has very good latency characteristics was within 2% of the overall average bandwidth observed across all configurations while in open page systems it was within 4% of the overall average bandwidth.

4.2.4 Impact of Row buffer Management Policy

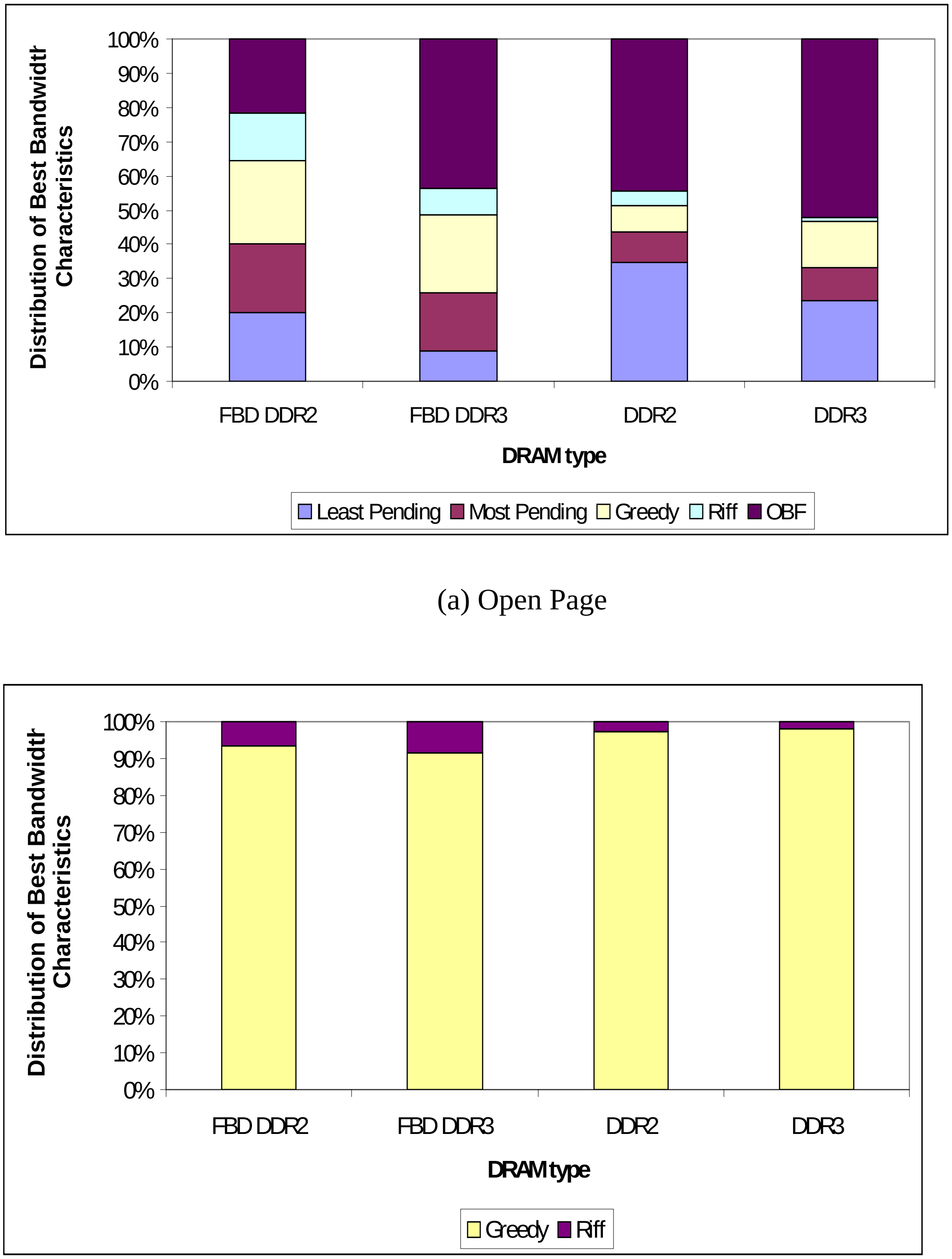
Typically FBDIMM configurations experienced higher average read latency than the corresponding DDRx configurations, but this effect was more pronounced in open page systems than in closed page systems. On the other hand, FBDIMM systems sustained as good or higher bandwidth than the DDRx system. Figures 4.23 and 4.24 show the improvement in latency and bandwidth respectively seen by using an FBDIMM system over using a DDRx system. Note that the data is plotted as a function of the bandwidth utilization of



(a) Open Page

(b) Closed Page

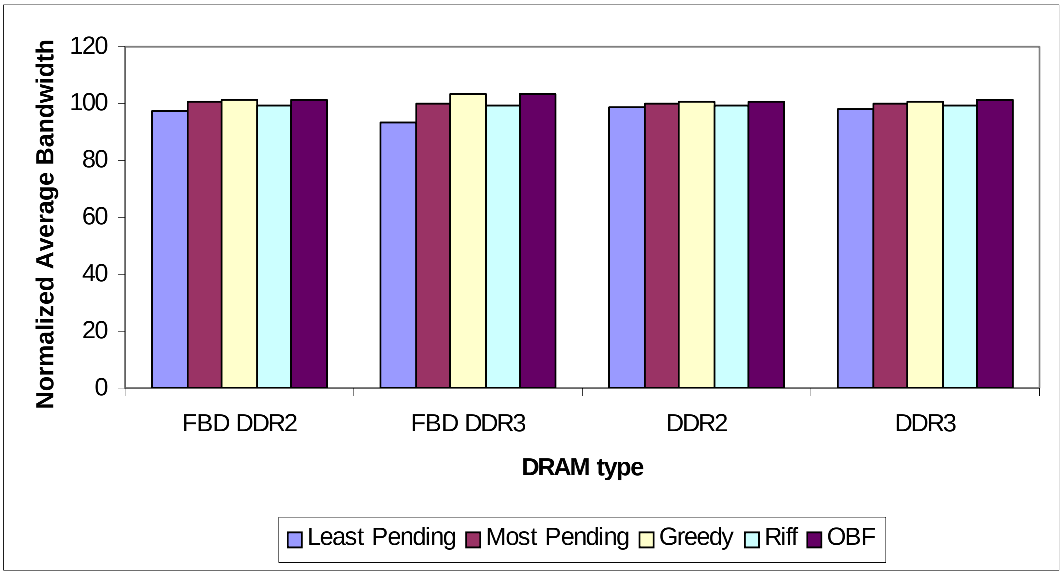
Figure 4.20. Normalized Average Read Latency for different DRAM types. The figure shows the distribution of which scheduling policy had the best latency characteristics across all the configurations studied. The data is shown for each DRAM type (plotted on the x-axis).



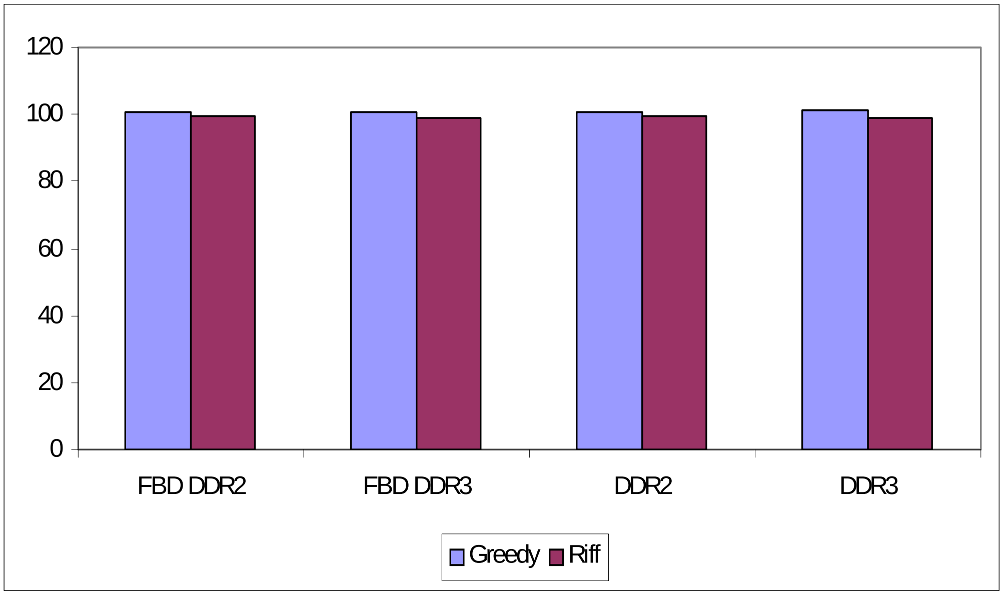
(a) Open Page

(b) Closed Page

Figure 4.21. Distribution of which scheduling policy has best bandwidth characteristics. The figure shows the distribution of which scheduling policy had the best latency characteristics across all the configurations studied. The data is shown for each DRAM type (plotted on the x-axis).



(a) Open Page



(b) Closed Page

Figure 4.22. Normalized Average Bandwidth for different DRAM types. The figure shows the distribution of which scheduling policy had the best latency characteristics across all the configurations studied. The data is shown for each DRAM type (plotted on the x-axis).

the workload. In an open page system, the access latency is reduced by taking advantage of locality in DRAM pages. In an FBDIMM open page system a side effect in the reduction of the access latency associated with DRAM row cycle times is that now the additional cost of serialization and daisy-chaining dominate the latency costs. Consequently, open page DDRx systems have significantly better latency values than their FBDIMM counterparts.

Another interesting phenomenon was that despite the heterogeneous nature of the workloads studied, there was significant row locality in the system (as seen in figure 4.25). As expected the bank hit rate i.e. the proportion of access that access a currently open DRAM row, drops as the number of threads in the workload increased. Thus, we see that a multi-threaded workload with 2 threads has more than double the row locality of the 8 threaded workloads.

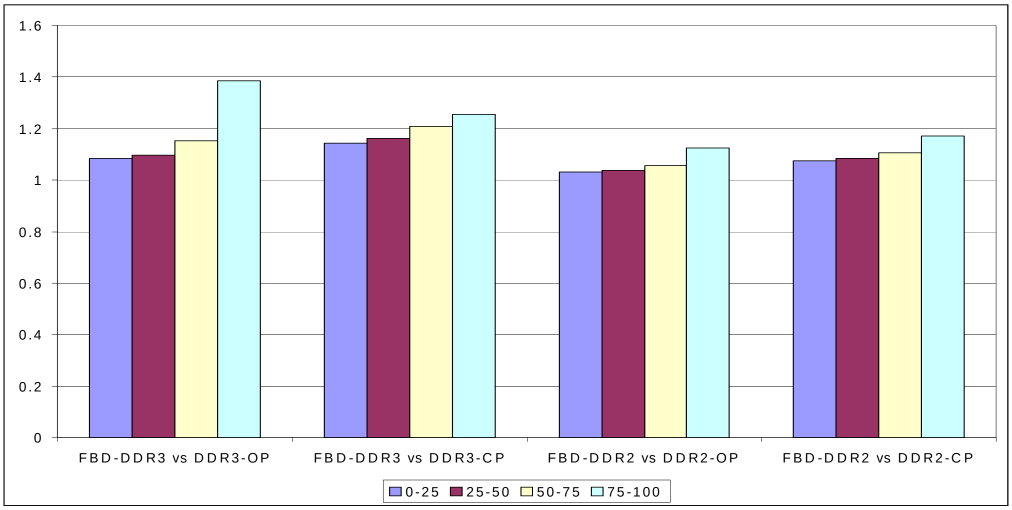
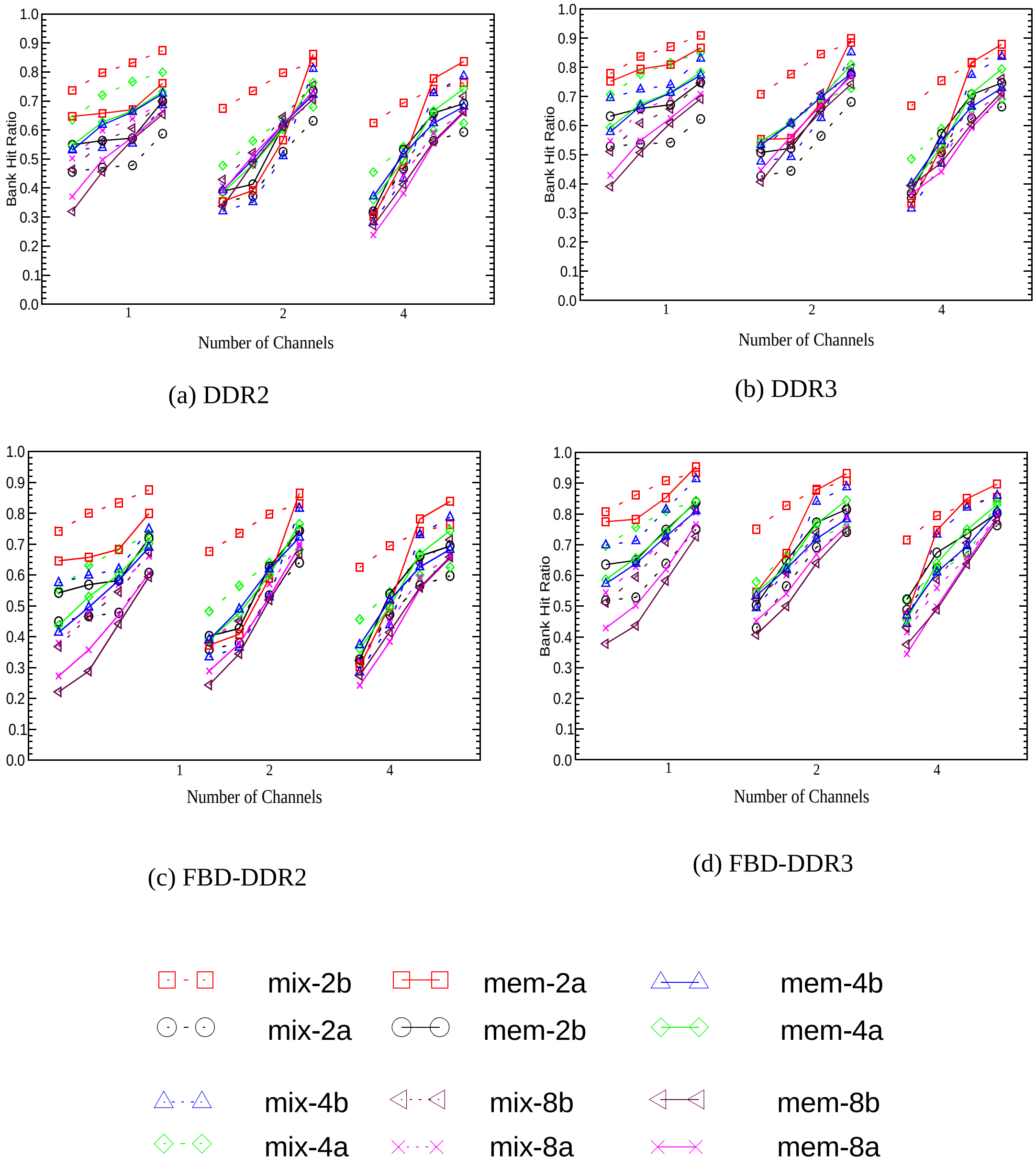


Figure 4.24. FBD vs. DDRx: Bandwidth Improvements for different row buffer management policies. The figure shows the improvement in bandwidth achieved by a FBD system over its corresponding DDRx system as a function of workload bandwidth utilization for open page and closed page systems. The bandwidth utilization classification is done using the observed bandwidth for the corresponding DDRx system. Note that the bandwidth improvements are calculated for all workloads studied and all configurations run and are shown normalized against the values seen for a DDRx system.



(a) DDR2

(b) DDR3

(c) FBD-DDR2

(d) FBD-DDR3

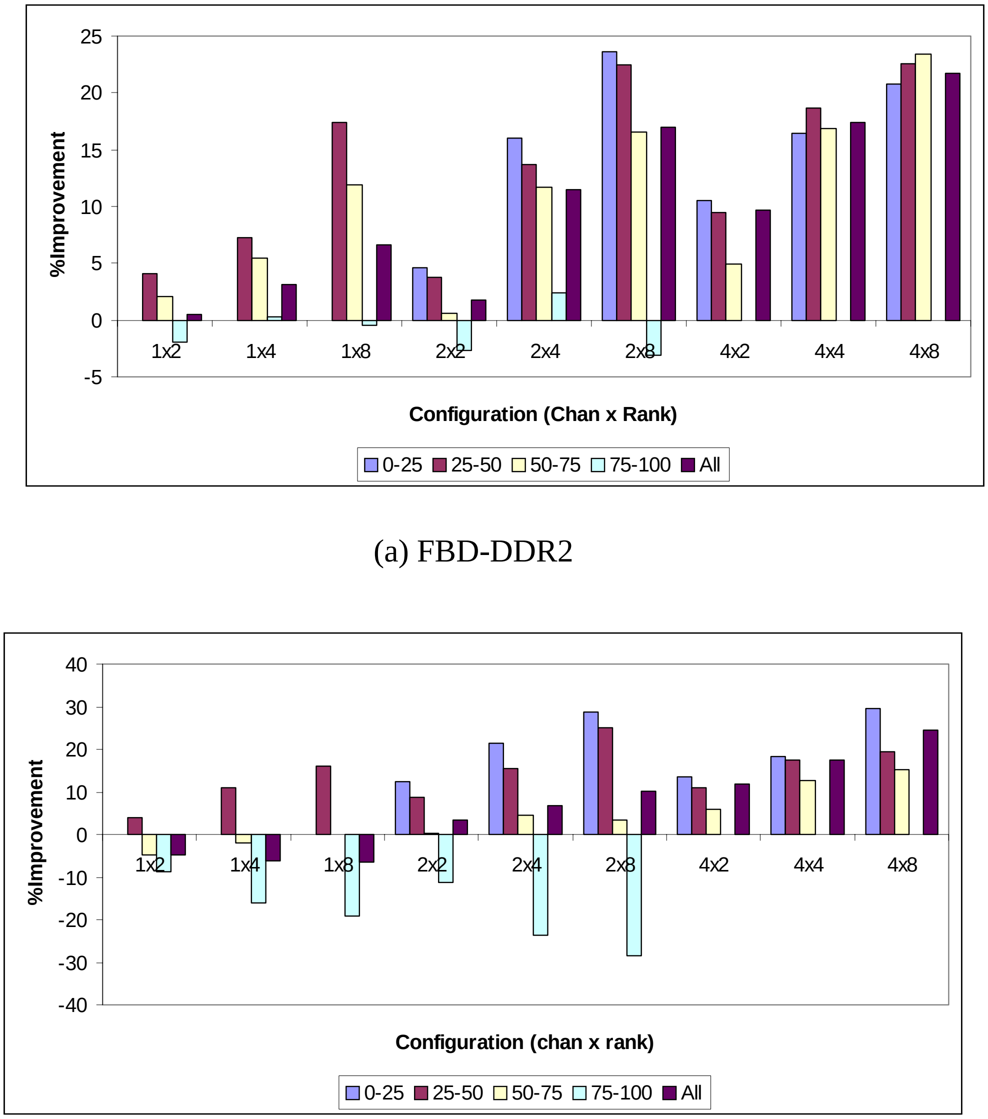
Figure 4.25. Average Bank Hit Ratios for various workloads. The graphs show the variation in average sustained bandwidth for a given configuration for the different dram types studied. All data is for a system which uses the open page row buffer management policy, does not use posted CAS and uses a burst length of 8. Each graph plots the average bandwidth in Gbps on the y-axis and the memory system configuration on the x-axis. On the x-axis the system configurations are grouped by the number of channels. Within each channel group, the number of ranks increases as you go from left to right. Note that the y-axis scales for the graphs are different.

4.2.5 Impact of Variable Latency Mode

The relative improvement in latency and bandwidth characteristics of a system using the variable latency mode setting over the fixed latency mode setting is a function of both the number of ranks in the channel and the channel utilization. Variable latency mode had better performance characteristics for light - moderately loaded systems, while the fixed latency mode did better in busier systems with longer channels.The latter was seen mainly in FBD-DDR3 system.

Variable latency mode is effective in lowering the average read latency experienced by a transaction in most system configurations (Fig 4.26). For nearly all cases, the improvements drop as the bandwidth utilization of the workload increases. In the case of some configurations with 8 rank deep channels, there is a latency degradation suffered when bandwidth utilization is more than 75%. Note that none of the workloads we studied, had greater than 75% bandwidth utilization in a 4 channel configuration. This latency degradation is typically seen in FBD-DDR3 systems, where the overheads of transmission i.e. the flight time of the packet down the channel, are larger than in FBD-DDR2 systems. The flight time of a packet is independent of the channel's operation frequency and increasing the channel frequency only increases the number of channel cycles it takes. Latency improvements seen by using variable latency mode are larger in open page systems than in closed page systems (Fig 4.27).

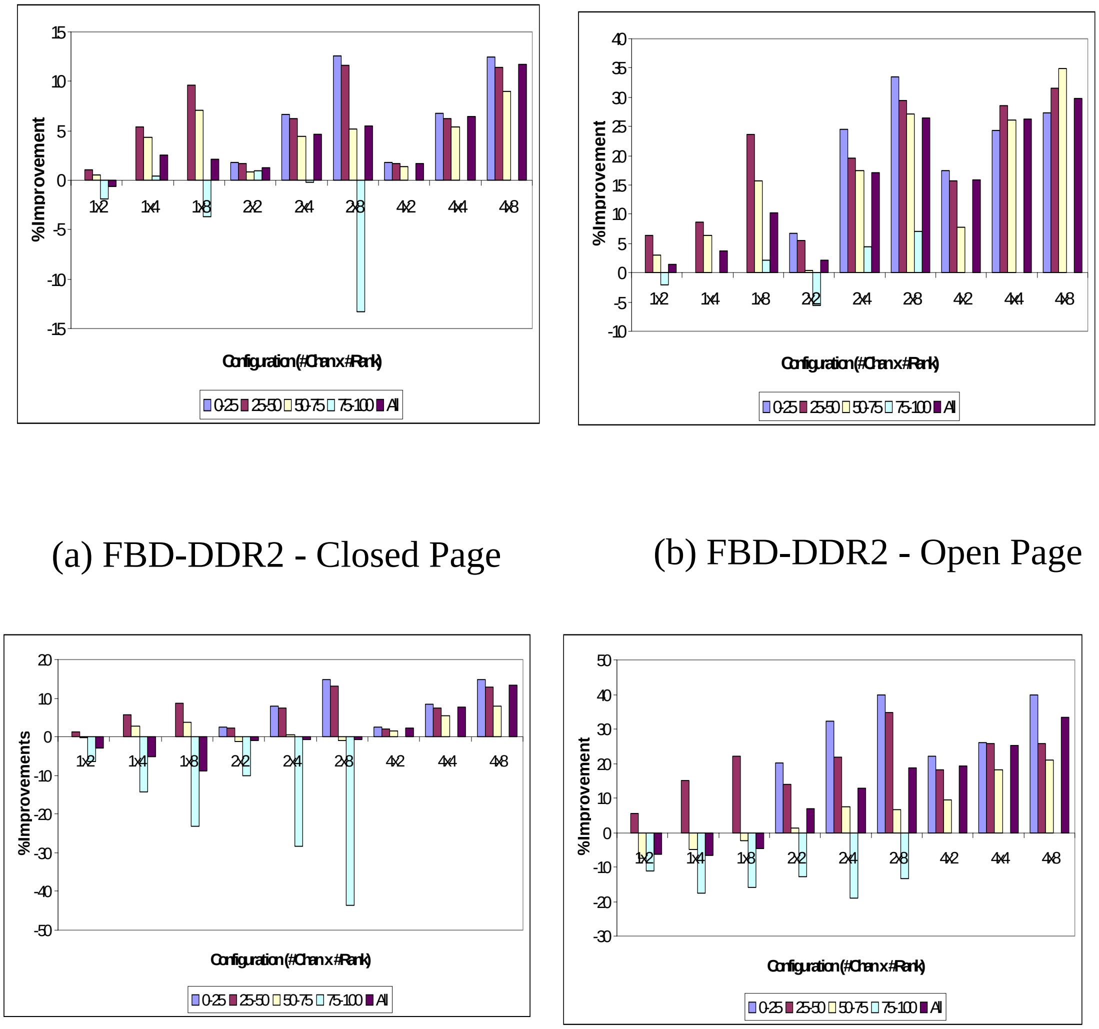
Figure 4.28 shows the improvements in total system bandwidth achieved by using the variable latency mode. Using the variable latency mode, especially at higher bandwidth utilizations ends up lowering the overall bandwidth in the system. This is because variable



(a) FBD-DDR2

(b) FBD-DDR3

Figure 4.26. Improvements in read latency from using Variable Latency Mode. The graphs show the improvements in average read latency of using variable latency mode over using fixed latency configuration for different memory system topologies. On the x-axis, the system configurations are grouped by the configuration, specified by the number of channels x number of ranks. Within each channel group, the percentage of bandwidth utilization increases as you go from left to right, with the rightmost bar representing the overall average latency improvement. Note that we plot the average improvement at a given system topology across all configurations run for the particular topology point.



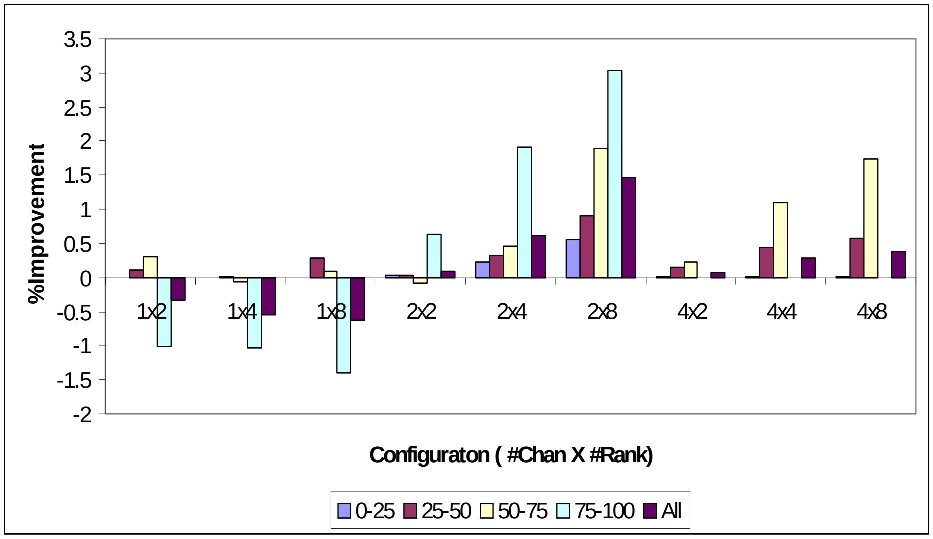
(a) FBD-DDR2 - Closed Page

(b) FBD-DDR2 - Open Page

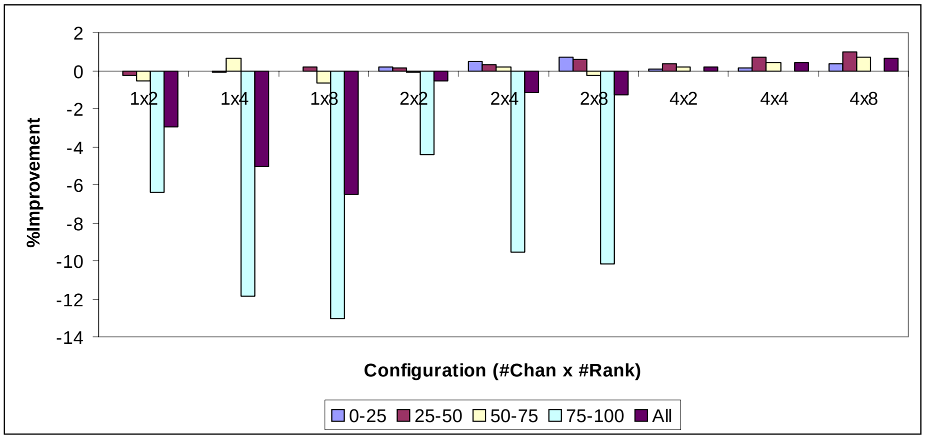
(c) FBD-DDR2 - Closed Page

(d) FBD-DDR3 - Open Page

Figure 4.27. Improvements in read latency from using Variable Latency Mode as a function of row buffer management policy. The graphs show the improvements in average read latency of using variable latency mode over using fixed latency configuration for different memory system topologies. On the x-axis, the system configurations are grouped by the configuration, specified by the number of channels x number of ranks. Within each channel group, the percentage of bandwidth utilization increases as you go from left to right, with the right-most bar representing the overall average latency improvement. Note that we plot the average improvement at a given system topology across all configurations run for the particular topology point.



(a) FBD-DDR2



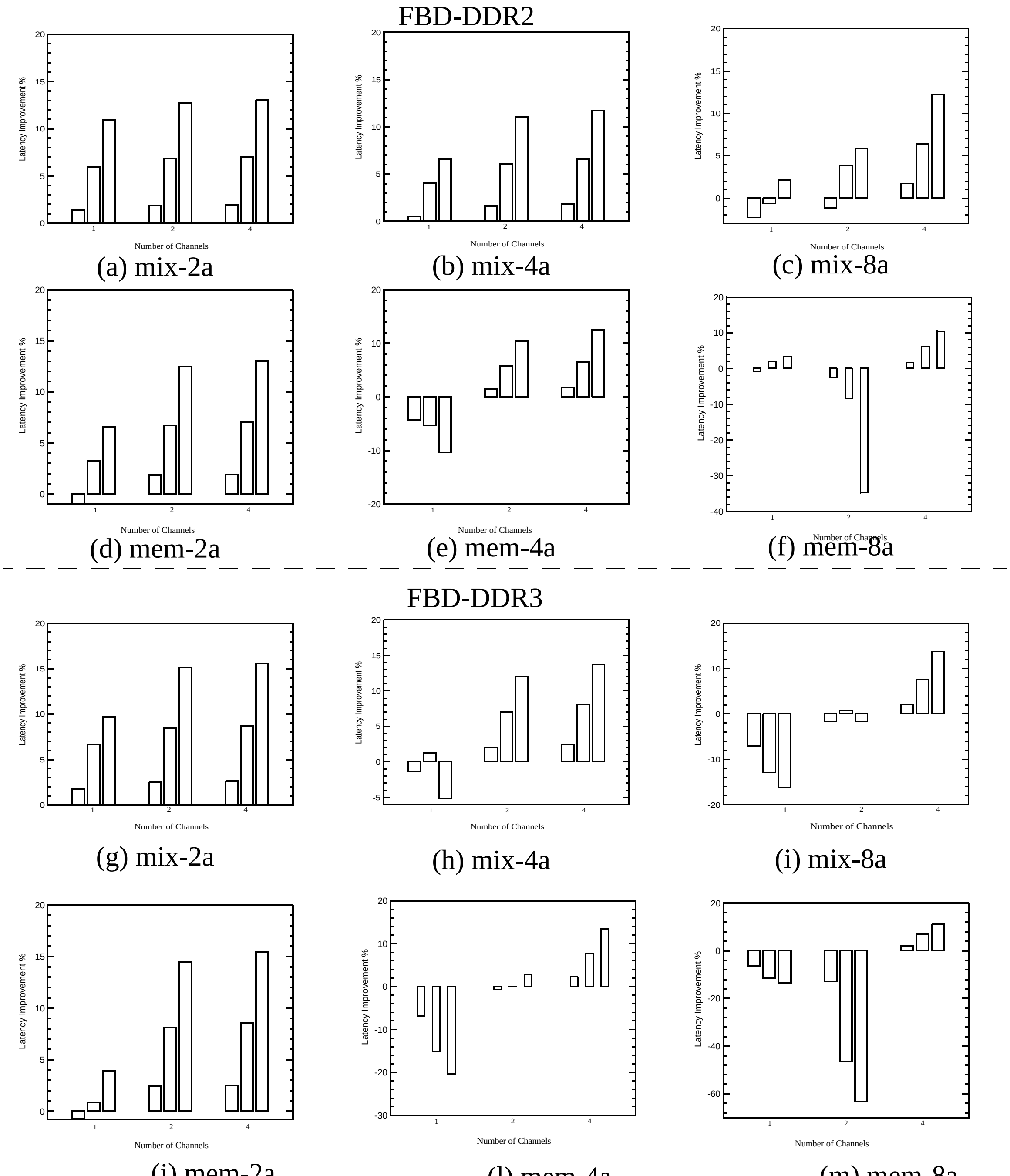
(b) FBD-DDR3

Figure 4.28. Improvements in bandwidth from using Variable Latency Mode. The graphs show the improvements in bandwidth of using variable latency mode over using fixed latency configuration for different memory system topologies. On the x-axis, the system configurations are grouped by the configuration, specified by the number of channels x number of ranks. Within each channel group, the percentage of bandwidth utilization increases as you go from left to right, with the right-most bar representing the overall average bandwidth improvement. Note that we plot the average improvement at a given system topology across all configurations run for the particular topology point.

latency mode reduces the ability of the memory controller to pack the return data channel tightly. At lower utilizations, the reduction in latency contributes to the ability of the memory controller to handle transactions more quickly, and consequently a slight increase in bandwidth is possible.

Figure 4.29 shows the average percentage improvement in latency obtained by operating the system in variable latency mode for different workloads. All numbers are given for a closed page system, but similar behavior was observed in an open page system as well. The variable latency mode was effective in most cases in lowering the average read latency. As expected, the variable latency mode is more effective for systems with deeper channels.

On the other hand in a heavily loaded system, using variable latency mode increases the average read latency. In variable latency mode, the read data latency for a transaction is dependent on the distance of the rank from the memory controller. Consequently using this mode lowers the effective utilization of the north link by introducing idle slots between read data returns from different ranks. As the channel depth increases, the duration of the idle gaps increases, making the latency characteristics worse (Figure 4.29 (c, e, f, i, l, m)). In variable latency mode, the round trip latency of a read is a function of the distance of the addressed DIMM from the memory controller. Consequently, a request to a further DIMM can be scheduled earlier than a read request to a DIMM closer to the memory controller. The latter request can consequently experience higher queueing delays waiting for the north link to become available. This too contributes to an overall deterioration in latency as


FBD-DDR2

(a) mix-2a

(b) mix-4a

(c) mix-8a

(d) mem-2a

(e) mem-4a

(f) mem-8a

FBD-DDR3

(g) mix-2a

(h) mix-4a

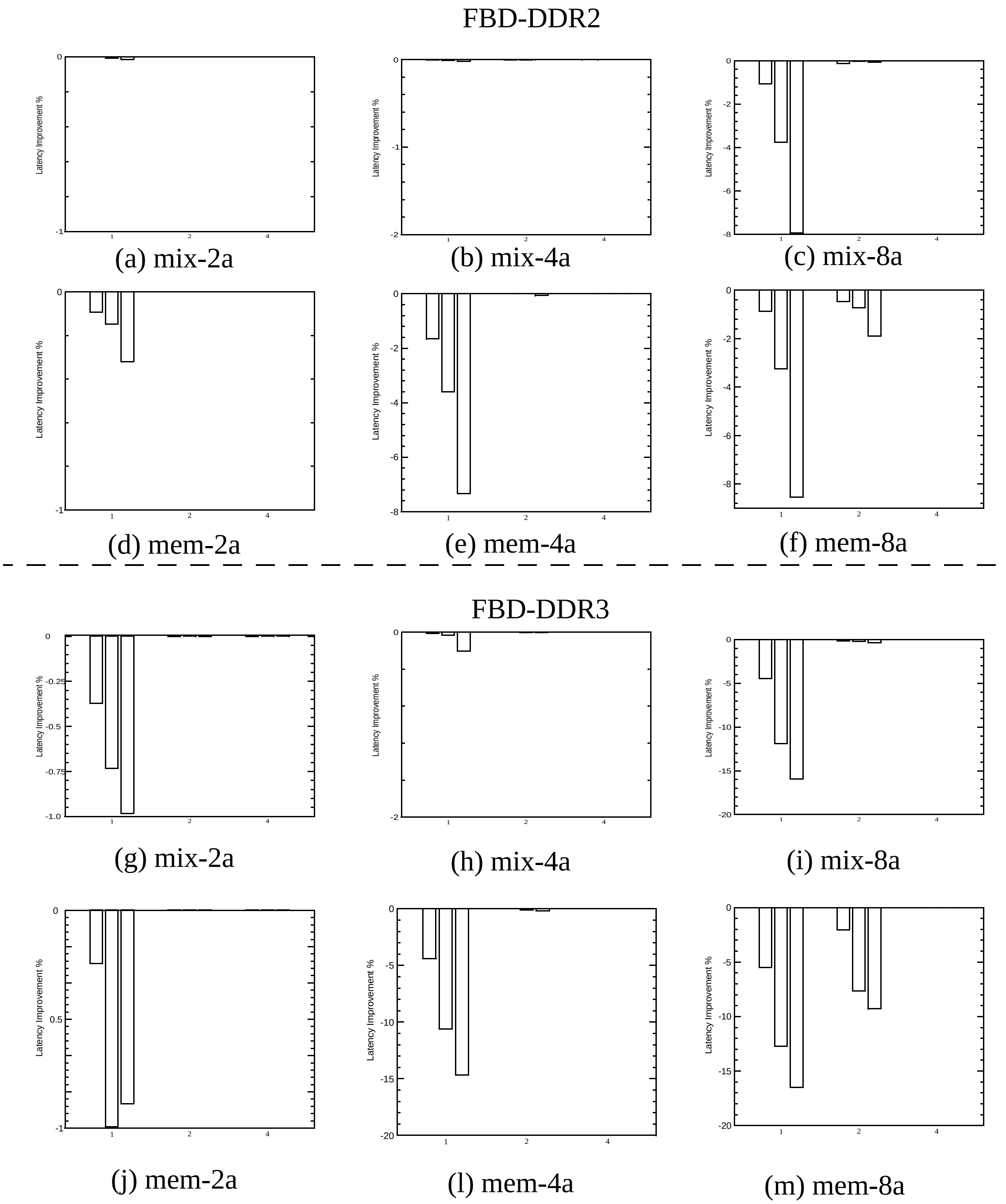
(i) mix-8a

(j) mem-2a

(l) mem-4a

(m) mem-8a

Figure 4.29. Improvements in read latency from using Variable Latency Mode. The graphs show the improvements in average read latency of using variable latency mode over using fixed latency configuration for different memory system topologies. On the x-axis, the system configurations are grouped by the number of channels. Within each channel group, the number of ranks increases as you go from left to right. Note that we plot the average improvement at a given system topology across all scheduling policies in a closed page system and do not use posted CAS. Graphs are for FBDIMM-DDR2 system while are for FBDIMM-DDR3 systems.


FBD-DDR2

(a) mix-2a (b) mix-4a (c) mix-8a

(d) mem-2a (e) mem-4a (f) mem-8a

FBD-DDR3

(g) mix-2a (h) mix-4a (i) mix-8a

(j) mem-2a (l) mem-4a (m) mem-8a

Figure 4.30. Improvements in bandwidth by using Variable Latency Mode. The graphs show the improvements in total observed bandwidth from using variable latency mode over using fixed latency configuration for different memory system topologies. On the x-axis, the system configurations are grouped by the number of channels. Within each channel group, the number of ranks increases as you go from left to right. Note that we plot the average improvement at a given system topology across all scheduling policies in a closed page system and do not use posted CAS. Graphs are for FBDIMM-DDR2 system while are for FBDIMM-DDR3 systems.

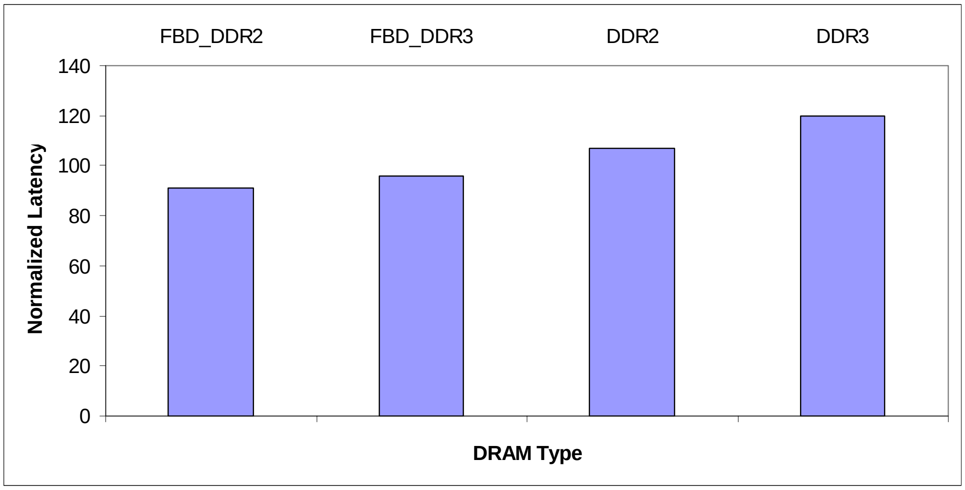


Figure 4.31. Latency Improvements by using posted CAS. The graphs show the improvement in latency by using posted CAS. The graph plots the normalized average latency on the y-axis and the dram type on the x-axis. The latency is normalized against the average latency (across all runs, configurations and workloads) for a system that does not use posted CAS.

the length of the memory channel increases and the number of outstanding requests increases.

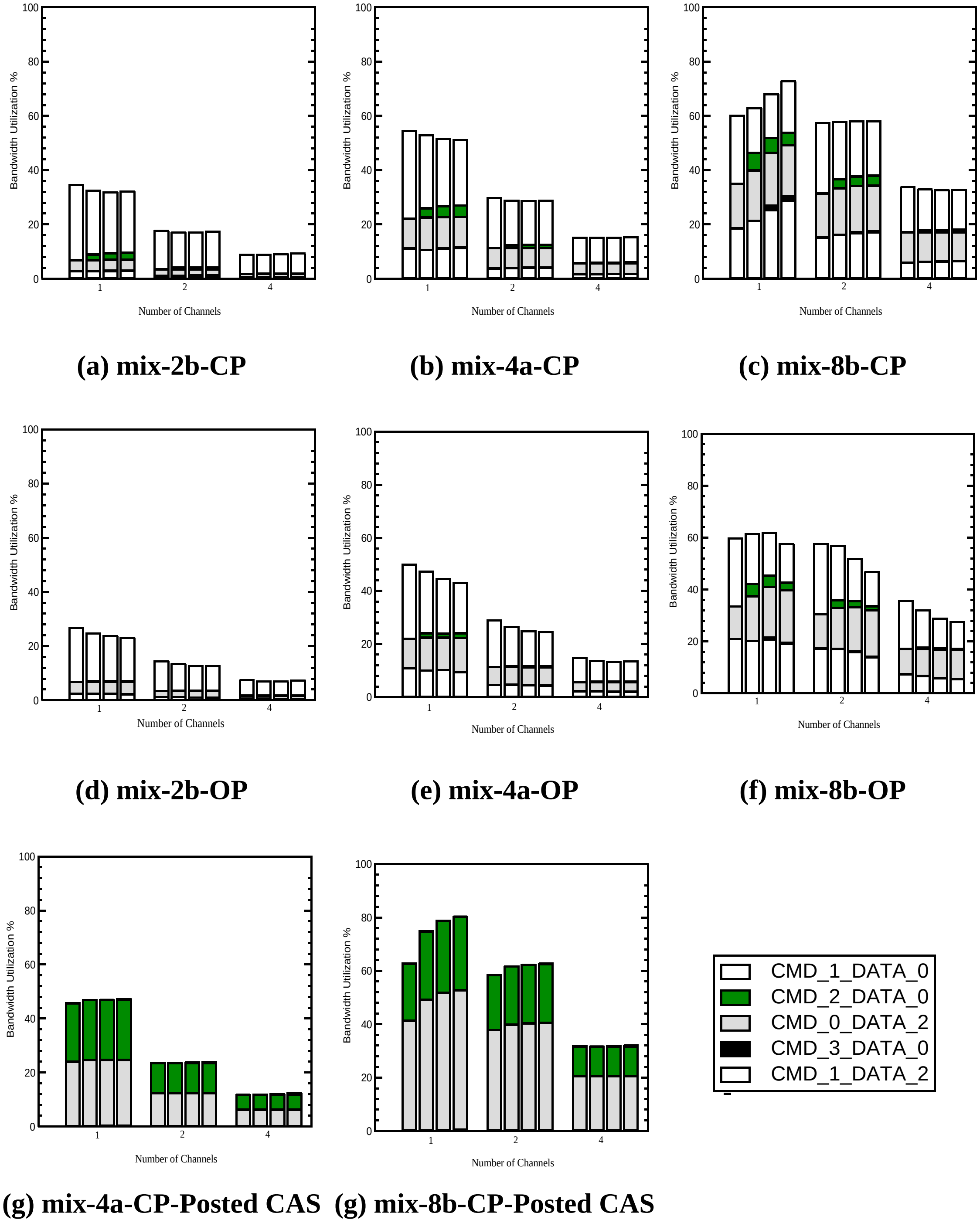
4.2.6 Impact of posted CAS

Posted CAS is a phenomenon where the memory controller can send a RAS and a CAS command to the DRAM in consecutive cycles. The DRAM devices delay the processing of the CAS command till a pre-specified additional delay time has been past. Posted CAS is typically used in closed page systems, where the DRAM command set per transaction is fixed and using posted CAS makes memory controller design simpler. To support posted CAS in a FBDIMM system, memory controllers typically send the RAS and CAS in the same command packet. The RAS is issued to the DRAM in the first DRAM command clock cycle while the CAS command is released to the DRAM in the next command cycle. The FBDIMM protocol supports this by design, since the command occupying the first command slot is issued to the DRAM in cycle *n*, while the commands occupying the other slots are released to the DRAM in cycle *n+1*.

The use of posted CAS results in a degradation of latency in a FBDIMM system, unlike in a DDRx system as compared to a system which does not use posted CAS. The reduction in the available southbound channel bandwidth for a FBDIMM system using posted CAS results in this increase in the overall read latency. By contrast, in a DDRx system, the use of posted CAS has no impact on the command bus utilization, and we see that the latency characteristics are better when using posted CAS. Sending the CAS separately from the RAS, introduces an arbitrary delay between the opening of the row and the data being sent back to the memory controller. This arbitrary delay contributes to an increase in the read latency of a transaction in a system not using posted CAS as opposed to one using posted CAS. In the case of FBDIMM systems, the unavailability of the southbound link in a system using posted CAS outweighs any delays introduced to the memory controller in a system which does not use posted CAS.

4.2.7 Southbound link utilization

The FBDIMM south link is shared by both commands and write data. The graphs in figure 4.32 provide the distribution of bandwidth utilization of the different types of FBDIMM frames. A command frame can carry up to three commands while a command-data frame carries 9 bytes of data (8-bytes of data and 1 ECC-byte) and one command. In the figure, command frames are further distinguished by the number of occupied command slots. The system does not send a data frame unless data is present thus there are no zero-



(a) mix-2b-CP **(b) mix-4a-CP** **(c) mix-8b-CP**

(d) mix-2b-OP **(e) mix-4a-OP** **(f) mix-8b-OP**

(g) mix-4a-CP-Posted CAS **(g) mix-8b-CP-Posted CAS**

Figure 4.32. Southbound bus utilization. The graphs show the utilization of the southbound bus bandwidth for various applications. Each graph plots the percentage bandwidth utilization on the y-axis and the memory system configuration on the x-axis. On the x-axis the system configurations are grouped by the number of channels. Within each channel group, the number of ranks increases as you go from left to right.

data frames. In the figure, graphs are shown only for a fraction of the workloads we studied, but we observed similar behavior for the other workloads as well.

As expected southbound bus utilization is a strong function of the number of threads in the workload with the 2 program workloads using around 30-40% of the southbound bandwidth and the 8 program workloads using as much as 80% of the south link bandwidth. For workloads with 2 or 4 programs, Fig 4.32(a, b), increasing the number of channels resulted in a linear decrease in southbound utilization, while for 8 program workloads which typically have larger bandwidth requirements, this decrease was observed only when the number of channels was doubled from 4 to 8, Fig 4.32(c).

Despite the ability to send multiple commands in a frame, the opportunities to do so are more limited in a system which does not use posted CAS. Across all benchmarks, we found that the memory controller is more likely to schedule a command by itself. For workloads with lower bandwidth utilization like mix-2a (Fig 4.32 a), the most commonly transmitted frame is the command frame with one DRAM command and padded with 2 NO-OPS. For workloads with larger write-traffic like mix-8a (Fig 4.32 (c)), the DRAM command is sent typically with a data payload. With increases in channel depth, the workloads see a slight increase in the number of command frames that have 2 valid commands. Regardless of the channel depth, less than 5% of the frames sent have 3 commands in them and less than 10% of the frames have 2 or more commands.

The command bandwidth used by an open page system is a function of the bank hit rate. A transaction in an open page system uses only one DRAM command when there is a hit to an open row and at least three commands (RAS, CAS and PRECHARGE) when there is a bank conflict. A closed page system on the other hand uses two commands, RAS and

CAS with implicit precharge for each operation. Any open page system that has less than 50% bank hit ratio will have a higher command utilization. Hence, we see for the mix-2b and mix-4a workloads which have bank hit rates of around 80% and 60% respectively that the southbound bus utilization is 2-10% lower in an open page system than in the closed page configuration (comparing Fig 4.32 (a) and Fig 4.32 (d), and Fig 4.32 (b) and Fig 4.32 (e)). In the case of the mix-8a workloads, the bank hit ratio increases from 40% in a 1- rank configuration to around 60% in a 8 rank deep configuration. We see that for the mix-8a workload, by comparing figures 4.32 (c, f), that the southbound bus utilization in the closed page configuration gradually increases as compared to that in the open page configuration, as you increase the number of ranks in the channel. In this case, the higher hit rate in the 8 rank deep configuration results in slightly lower command bandwidth requirements for the open page case.

Another memory controller parameter which has impact on the southbound bus utilization is the use of posted CAS. The use of posted CAS increases the southbound bus utilization by 10 to 20% (Fig 4.32 (g.h)) over the same configuration that does not use posted CAS (Fig 4.32 (b, c)). The effect of using posted CAS is seen in channels which have more than one rank because it reduces the ability of the memory controller to pack commands with data packets. Nearly all frames transmitted in a system which uses posted CAS carry at least one no-op.

4.3. Summary of Results

Our study indicates that an FBDIMM system provides significant capacity increases over a DDRx system at comparable performance. Although an FBDIMM system experiences an overall average latency increase compared to the corresponding DDRx system, we found that by efficiently exploiting the extra DIMM-level parallelism in a busier system the latency costs can be significantly lowered and even eliminated. The split-bus architecture makes the performance of the system sensitive to the ratio of read to write traffic. For instance, it enables workloads with both read and write traffic to sustain higher bandwidths than in a DDRx system. Workloads that use the system heavily tend to record higher gains because they tend to use both channels simultaneously.

Overall FBDIMM system had an average of 27% higher latency which were mainly contributed by workloads with bandwidth utilizations of less than 50% total DDRx DRAM bandwidth. This latency degradation become a latency improvement of nearly 10% for FBD-DDR3 systems as the application bandwidth utilization increased past 75% due to the ability of the FBD memory controller to use the additional DRAM level parallelism, split bus architecture and ability to send multiple DRAM commands in the same clock cycle. However, the additional system bandwidth available in a FBDIMM system resulted in an average of approximately 10% improvement in overall bandwidth with most of the benefits coming again for workloads with higher bandwidth utilization.

Even more interestingly the behavior of an application in the conventional memory architecture and the newer one are similar. The scheduling policies and row buffer management policies used in DDRx systems continued to perform comparably in FBDIMM systems. In both cases, a scheduling policy that prioritizes read traffic over write traffic had the best latency characteristics for both open page and closed page systems. A scheduling policy that prioritized traffic to currently open banks in the system had the best bandwidth characteristics while a greedy approach did very well in closed page systems. The performance of open page systems was sensitive to the distance between accesses to the same

DRAM row and the types of the requests. Many of the scheduling policies, such as open bank first, most pending etc., attempt to exploit locality in the DRAM rows to lower latency. We found that in addition to all these factors, a controller policy that prioritizes reads over writes is more effective. A designer who implements such a policy should take care to give higher priority to outstanding write transactions that delay read transactions to the same bank and to periodically drain write requests to prevent the memory transaction queue from filling up.

One difference that we found with regard to FBDIMM and DDRx system behavior was there response to the use of posted CAS, a DRAM protocol feature which simplifies memory controller design by allowing the memory controller to bundle a row activation and read/write command in back-to-back command cycles. Unlike DDRx systems, FBDIMM systems using posted CAS had worse latency and bandwidth characteristics than systems which did not use this. This difference arose from the organization and use of the FBDIMM command frame and the sharing of the FBDIMM southbound bus by commands and write data.

Detailed measurements of the contributors to the read latency of a transaction revealed that a significant contributor was delays associated with the unavailability of memory system resources like the southbound channel, northbound channel and DRAM. Scaling the memory system configuration, by adding more ranks or channels, resulted in the unavailability of each of these factors varying in a different fashion and interacting in different ways to impact the observed latency. In general we observed that short channel FBDIMM systems are limited by DRAM availability, while long channel FBDIMM systems are bound by channel bandwidth. This problem is exacerbated in variable latency mode configurations where significant latency and bandwidth degradation occur due to inefficient usage of the northbound FBDIMM channel.

While the variable latency mode is effective in lowering the latency of a read transaction in a system by 2.5-20% in lightly loaded systems, the exact opposite effect was observed at higher utilizations. This suggests that a controller that dynamically switches the mode of operation in response to changes in system utilization would see less performance degradation.

Chapter 5: Optimizations for Variable Latency Mode

In the earlier chapter, we demonstrated that the average read latency of the FBDIMM channel can be improved by operating the channel in the variable latency mode in some cases. In this chapter, we further examine how to improve the performance of the variable latency mode, especially in cases where its use resulted in a deterioration in latency over the fixed latency mode. Two main techniques to improve the channel usage are studied in detail, the first is to allow read data to return in a different order than it was issued in and the second to allow read data frames to be buffered at the AMB before being burst back to the memory controller.

5.1. Performance Characteristics of Variable Latency Mode Systems

5.1.1 Limit Study

One of the commonly used approaches to gauge the performance of a memory system protocol is to conduct a limit study using random-address traces as input. The latency and bandwidth values are measured for a random address trace whose input arrival rate is varied. The latency typically gradually increases till a particular bandwidth value after which it dramatically increases. This point on the latency-bandwidth curve represents the maximum sustainable bandwidth of the system prior to its getting overloaded. Memory controller design is focussed on moving this latency-bandwidth curve to the right i.e. improving the read latency values at higher bandwidth values.

For this study, we use the memory system parameters specified in table 5.1. Note that the memory controller model used is similar to that one used described in 4.1.1. The main differences are that in this section, we assume an infinitely large BIU i.e. transactions are

TABLE 5.1. Memory System Parameters

Parameter	**Value**
DRAM Technology	FBDIMM - DDR3-1333-8-8-8
Queue Size	16 Read/16 Write
Paging Policy	Closed Page
DIMMs/DIMM	1
# DIMMs	2,4,8
# Channels	1 Logical Channel
# Banks	8
Burst Length	4,8
Posted CAS	Enabled
Refresh	Disabled

placed in the BIU as soon as they are available and scheduled to the memory controller only when a spot in the transaction queue is available. The system uses split transaction queues i.e. transaction queues which are split based on the type of transaction.

Random Address Traces. These input traces are generated using a random number generator. The input address stream is modelled as a poisson arrival process where each request is independent of the previous request. Each random address stream is identified using the following parameters

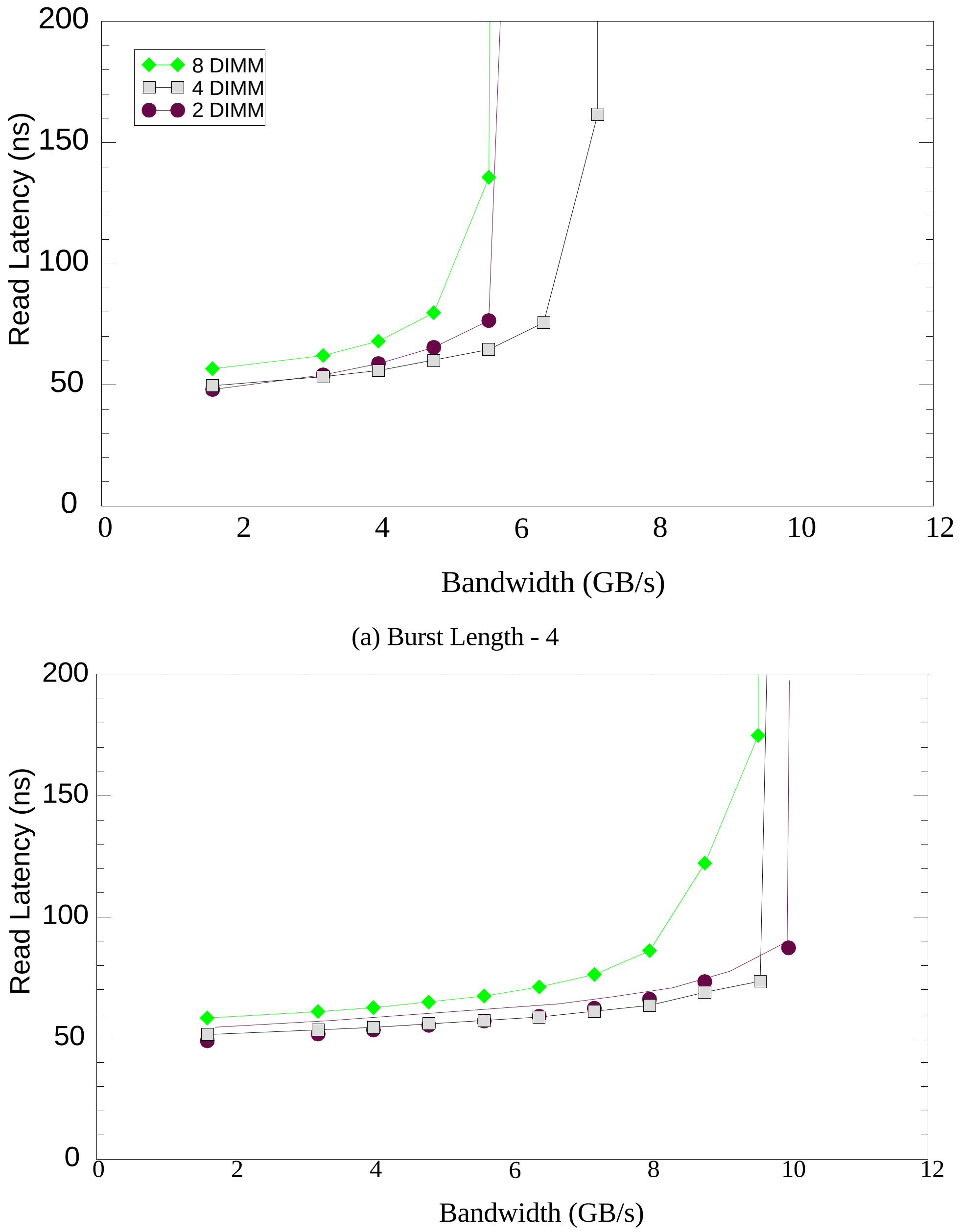
- Average issue bandwidth is the average arrival rate for the poisson process used to model the trace.
- Number of DIMMs in the system
- Proportion of read traffic, and

- Spatial Locality. The spatial locality is used to build input traces with and without hotspots.

5.1.1.1 Latency-Bandwidth Characteristics

This section first gives an overview of how the performance of a FBDIMM system varies with system configuration parameters such as channel depth and burst length. The maximum sustainable bandwidth obtained from a system is limited for systems with fewer DIMMs by conflicts for the same DRAM resources and for systems with deeper channels, i.e. more DIMMs in the channel, by inefficiencies in north link usage. The in-order return of read data prevents the scheduler from taking advantages of any idle gaps on the north link which may be present prior to the return of any previously scheduled read data. This can make the peak sustainable bandwidth of a 8-DIMM deep system lower than that of a 2-DIMM deep system. Burst length 8 systems are able to achieve greater maximum bandwidths due to the larger size of data transferred per request.

Figure 5.1 shows how the latency-bandwidth characteristics of the system vary with burst-length and channel depth i.e. numbers of DIMMs in the channel. Note that in all cases, the latency of a read transaction increases gradually with system throughput. This latency trend holds till a certain throughput value at which point the latency increases dramatically. This dramatic increase in latency occurs when one or more of the various required resources i.e. DIMM, links become a bottleneck. The system throughput at this point, also referred to as the knee of the latency-bandwidth curve, is the maximum sustainable bandwidth for the particular system.



(a) Burst Length - 4

(b) Burst Length - 8

Figure 5.1. Latency-Bandwidth Characteristics for base-line FBDIMM system. The latency-bandwidth characteristics are shown for different channel depths - 2, 4 and 8 deep channels, and different burst lengths - 4 and 8. The input for these curves is a random-generated address trace which is equally likely to be addressed to any DIMM in the channel. The input comprises of 66% read traffic. The x-axis plots the sustained bandwidth in Gbps while the y-axis plots the read latency in ns.

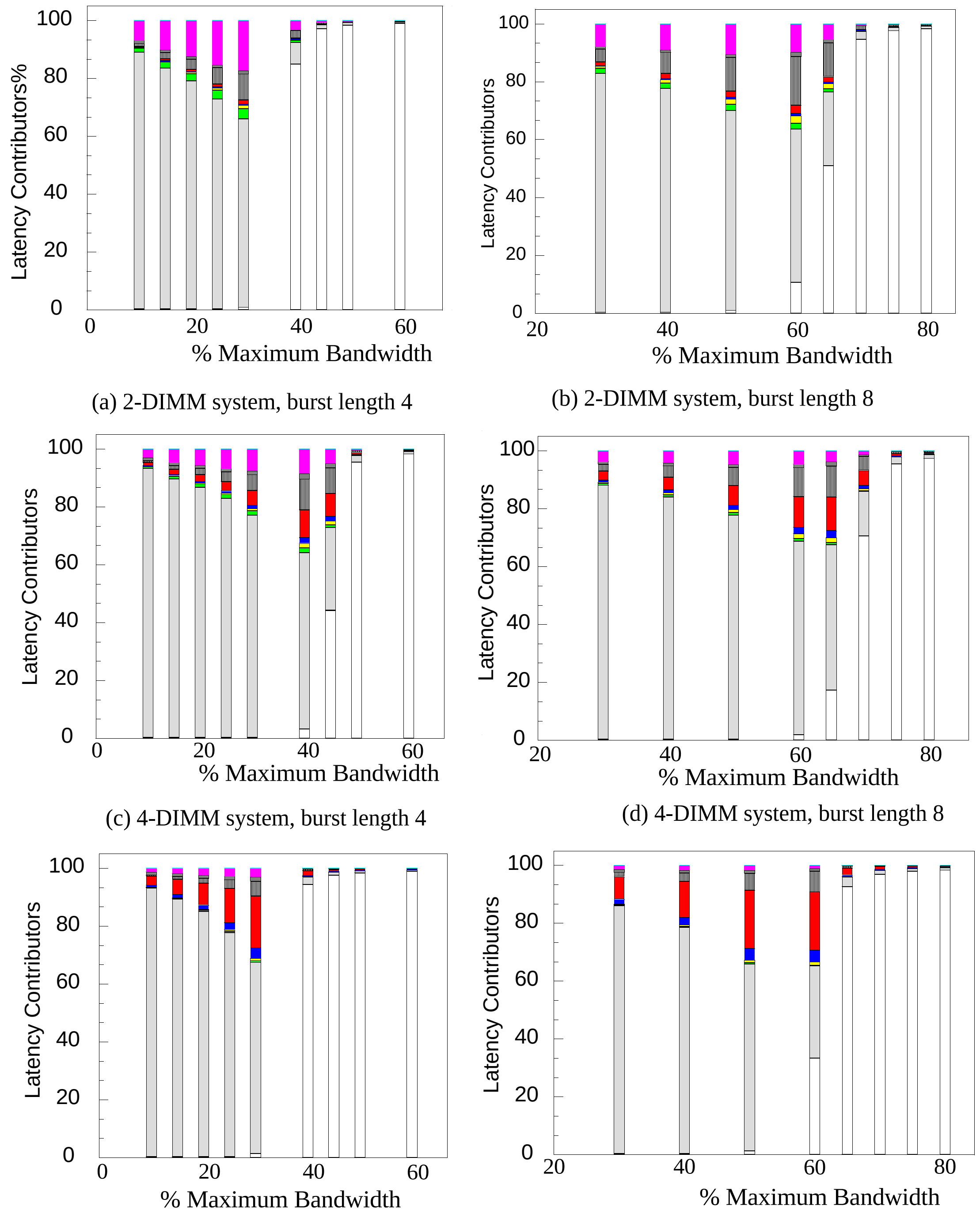
Increasing the number of DIMMs in the channel increases the DIMM-level resources in the system, thereby lowering the queueing delay due to the unavailability of the DIMM. Simultaneously more DIMMs in the channel increases the default read latency due to the additional serialization. Latency also increases due to the increased unavailability of the link due to the inefficiencies in link utilization. These inefficiencies arise because the memory controller scheduling algorithm is unable to effectively schedule the return link. In order return constrains transactions to return read data in exactly the order that it was scheduled in. This means that upstream DIMMs must forgo opportunities to utilize idle slots in the channel when a transaction to a downstream DIMM is in progress. This effect is more pronounced in longer channels because of the larger disparity between the round trip time for transactions going to the first and last DIMM in the system. This lowers the performance of the system as more DIMMs are added to the channel. The interaction of these three different factors determine how the performance characteristics of the system changes with the addition of DIMMs in the system.

The DRAM is the main bottleneck for systems with fewer DIMMs. Adding more DIMMs helps alleviate this bottleneck. When going from a 2-DIMM deep channel to a 4-DIMM deep channel, the gains due to DIMM-level parallelism can successfully offset the increased cost of serialization such that there is an improvement in the latency-bandwidth characteristics of the system as seen in Fig 5.1. On the other hand, further increases in the length of the channel, i.e. going from a 4-DIMM to an 8-DIMM channel, results in a lowering of system performance. This is because the costs of increased serialization and the inefficiencies of link utilization outweigh the gains due to additional DRAMs in the system.

The additional DIMM level parallelism results in an increased competition for the links as well further worsening the queueing delay waiting for the links to become available.

Burst length 8 systems sustain higher bandwidths than burst length 4 systems for a given channel configuration (figures 5.1(a) and 5.1(b)). A burst length 4 system require twice the number of transactions to achieve the same bandwidth as a burst length 8 system. This makes burst length 4 systems more sensitive to DIMM-level constraints such as row-to-row activation time, row cycle time and the four-bank-activation window.

Fig 5.2 shows how the chief contributors to latency for a FBDIMM system vary as the arrival rate of transactions given in terms of a percentage of the total data bandwidth in the system is increased. The average read latency is divided into two main components the queueing delay overhead and the transaction processing overhead. The queueing delay component refers to the duration the transaction waited in the queue for various resources to become available. These resources include the memory controller request queue, the south link, the DIMM (including on-DIMM command and data buses and bank conflicts), and the north link. Multiple resource unavailability are also monitored and reported. Note that if the memory controller transaction queue is unavailable, then we do not take into account its overlap with any other queueing delay component. The processing component is the default transaction processing cost. In general we observed that at the point when the latency of the system increases rapidly the queueing delay associated with unavailability of the memory controller queues becomes a substantial portion of the overall latency. The chief contributor to the queueing delay at this point is the main bottleneck of that particular



(a) 2-DIMM system, burst length 4

(b) 2-DIMM system, burst length 8

(c) 4-DIMM system, burst length 4

(d) 4-DIMM system, burst length 8

(e) 8-DIMM system, burst length 4

(f) 8-DIMM system, burst length 8

Figure 5.2. Latency Contributors. The graphs show the contribution of various sources to the overall read latency of a transaction for a given system operating at a certain issue bandwidth point. Note that all contributions are given in terms of percentage. The actual latency values seen are different. The system is past its acceptable latency operational point when the latency overhead due to the transaction queue being full emerges

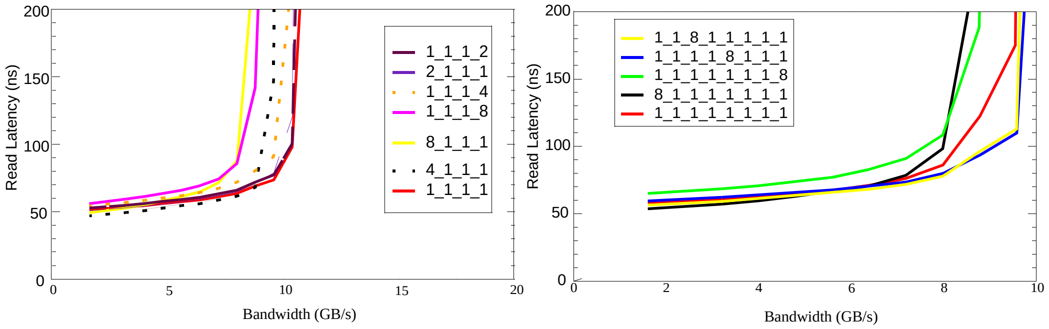
configuration. Note that the latency values are all normalized to 100%. Note that unavailability is measured every channel clock cycle and not every scheduling cycle.

From figures 5.2 (a) and 5.2 (b) we can see that in a 2-DIMM system, the DIMM-unavailability contributes to the bulk of the queueing delay. Adding additional DIMMS, reduces the DIMM-associated bottleneck but the links start emerging as a bottleneck. For a 8-DIMM system, both burst length-4 and 8, figures 5.2 (c, d), it can be see that the queueing delay for a transaction is largely due to unavailability of both the links. The link unavailability increases due to increased usage and the north link unavailability increases mainly due to the inefficient usage of the link described earlier.

Impact of spatial locality . The impact of spatial locality in the input stream is a function of the intensity of the hot-spot, the DIMM which is the hot-spot, the number of DIMMs in the channel and the burst length. For systems with 4 or less DIMMs, hot-spotting lowers the maximum sustainable throughput of the system due to increased pressure on the DRAM. This is true in a 8 DIMM deep channel when lower proportion of the transactions are destined to a particular DIMM. As the proportion of traffic is increased the opposite effect is observed i.e. the maximum sustainable bandwidth obtained from a stream with spatial locality is better than one without. This occurs when the locality in the stream helps improve the link utilization sufficiently so that the gains due to better use of the link offset any reductions in performance due to DRAM resource unavailability. Inputs with hot-spots at the first and last DIMM have lower maximum sustainable bandwidth than those with hot-spots located at other DIMMs.

The graphs in Fig 5.3, show how the latency-bandwidth characteristics of the system are impacted by spatial locality in the input stream. Hot-spotting is achieved by shaping the randomly generated input trace appropriately. Each input traffic pattern is labelled using a sequence of numbers (one for each DIMM in the system)[1] and each number represents the proportion of traffic addressed to the particular DIMM. The per-DIMM traffic proportion sequence is ordered from the closest DIMM to the furtherers DIMM.

The exact impact of hot-spotting on system performance is a function of the degree of hot-spotting i.e. the proportion of transactions, the location of the hot-spot i.e. which DIMM in the chain, the number of DIMMs in the channel as well as the burst length. For a



(a) 4- DIMM system, burst length 8

(b) 8 DIMM system, burst length 8

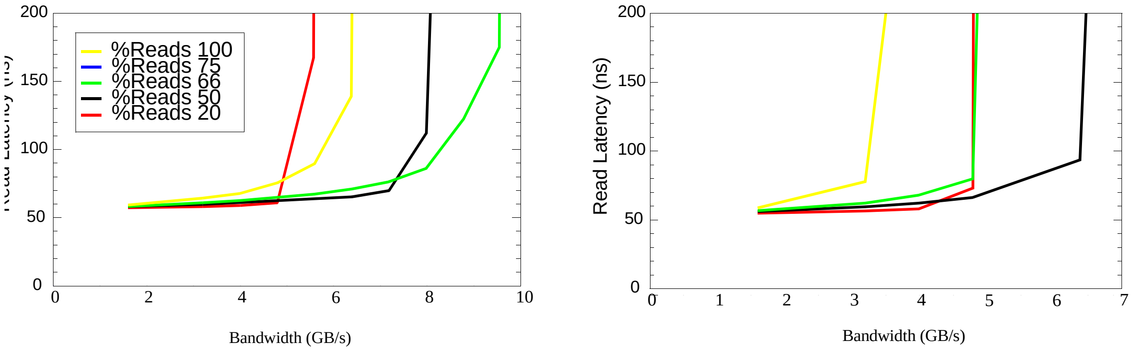
Figure 5.3. Impact of hot-spotting on latency-bandwidth. The figures show the impact of hot-spots in the input on the latency-bandwidth characteristics of the system. Each line in the graph represents input traffic with a different distribution across the DIMMs. The x-axis plots the sustained bandwidth in Gbps while the y-axis plots the read latency in ns. The input distribution is labelled by the proportion of traffic targeted at each individual DIMM in the system. For instance an input which has 4 times the traffic to the 2nd DIMM and equal proportions to other is labelled as 1_4_1_1.

[1] The spatial distribution is specified by the following nomenclature - < proportion of transactions to DIMM 0>_ < proportion of transactions to DIMM 1>. < proportion of transaction to DIMM n>. The DIMMs are ordered from 0 to n, where n is the total DIMMs in the systems minus one and DIMM 0 is the DIMM closest DIMM to the memory controller and DIMM n-1 is the DIMM furthermost from the memory controller. This proportion is used to determine the probability of a given transaction addressing the particular DIMM in question.

system with 2 and 4 DIMM deep channels, we found that the latency-bandwidth characteristics of the various distributions were all bounded by that of an input pattern which was equally likely to address any DIMM in the chain. Hot-spots lowered the maximum throughput possible for a given configuration by increasing the number of transactions in a stream which are delayed by DIMM-level constraints. Increasing the degree of hot-spotting i.e. sending more transactions to a particular DIMM further reduced the maximum sustainable bandwidth.

Hot-spots at the first and last DIMM in the chain have lower maximum sustainable bandwidth than traces with hot-spots at any other DIMM in the chain. Transactions which are addressed to the first DIMM in the chain experience the highest queueing delay due to link unavailability. This is because they are impacted by the link usage of all scheduled transactions. On the other hand, transactions which are destined for the last DIMM in the chain, delay the completion of transactions to every other DIMM in the chain. Consequently hot-spots to these locations end up lowering the achievable bandwidth in the system.

Fig 5.3 (b) shows that in an eight DIMM deep channel, when the degree of hot-spotting increases sufficiently that the latency-bandwidth curves are better than those seen in the case of an input stream with no hot-spots. Although spatial locality increases the conflicts for DIMM-level resources, it improves the overall efficiency of the northbound link. This latter phenomena is sufficient in an eight-DIMM deep channel to cause the latency-bandwidth curves of inputs with some degree of hot-spots to be better than a distribution that is equally distributed across all DIMMs.



(a) #DIMMs 8 Burst Length 8

(b) #DIMMs 8 Burst Length 4

Figure 5.4. Impact of proportion of read-traffic on latency-bandwidth. The figure shows how the latency bandwidth curve varies as the proportion of read traffic is varied. The x-axis plots the sustained bandwidth in GBps for different read-write ratios and y-axis shows read latency in nano-seconds.

Impact of varying proportion of read-traffic. A consequence of the split-bus architecture is that the performance of the system is sensitive to the ratio of read to write traffic in the input stream. The FBDIMM bandwidth is divided so that it provides as much read bandwidth and half the write bandwidth as a DDRx system. For input streams with smaller proportion of read traffic, the competition between write-data and commands results in the south link being a bottleneck resulting in a degradation in system performance. As the percentage of read traffic, we see that this bottleneck eases and the latency-bandwidth characteristics improve. For the systems with a significant proportion of read traffic, the north link is a bottleneck and this degrades system performance.

In general the best bandwidth can be obtained from a system with a read to write traffic ratio which matches the read to write bandwidth ratio. The only exception to this case is a system with 8 DIMM deep channel and a burst length of 4 (Fig 5.4 (b)). In this case the stream with 50% read traffic out-performs a system with 66% read traffic. In this particular case the inability to efficiently use the north link results in an input stream with 66% read traffic having a lower peak bandwidth than that of an input stream with 50% read traffic.

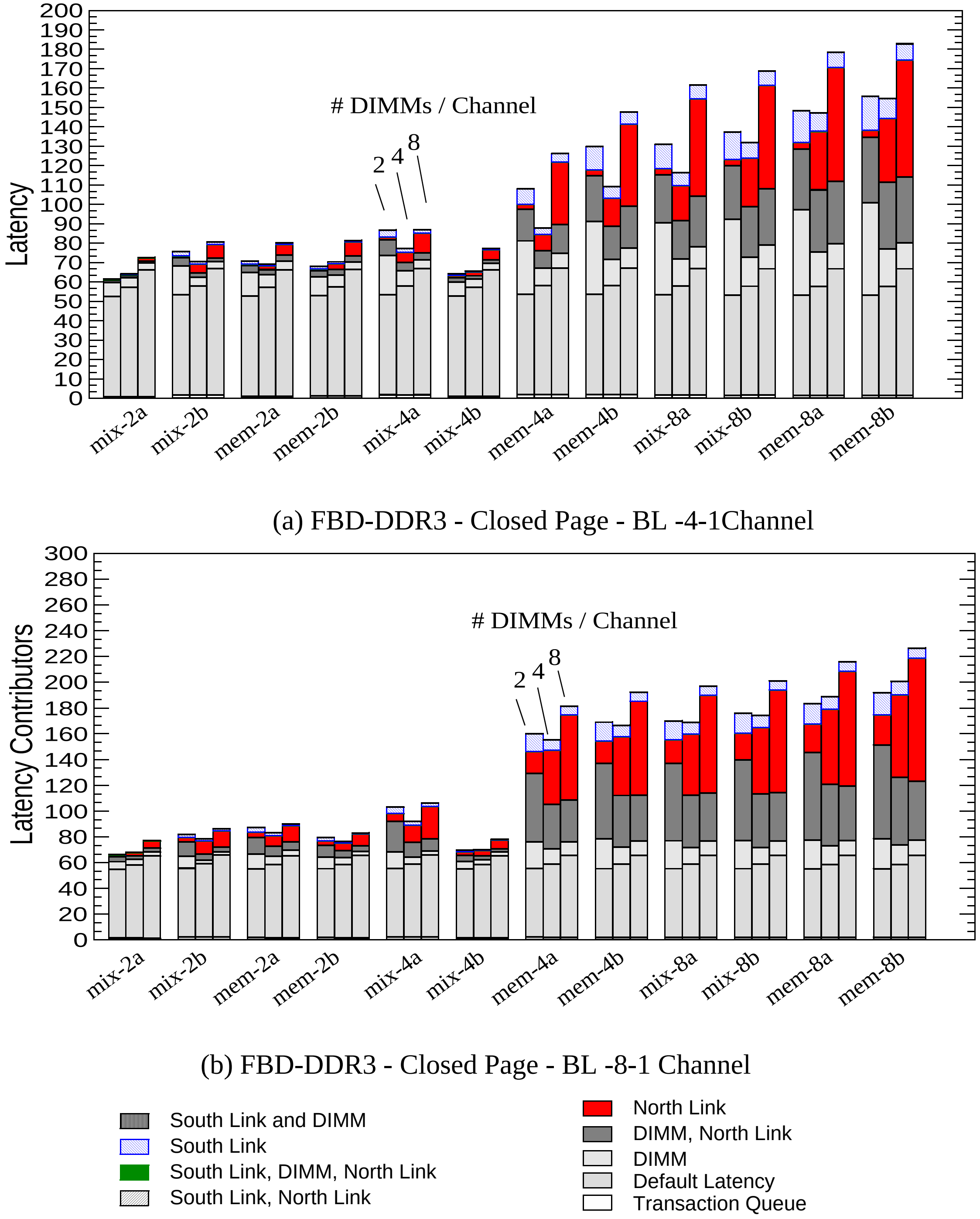
5.1.2 Application Performance Characteristics

In this section, we look at how the latency and bandwidth characteristics varied for various workload traces. We found that as in the case of the random address traces, applications benefit from the addition of DIMMs in the channel due to the reduction in contention of DRAM resources. This benefit lowers latency for applications when it outweighs the delays due to the increases in latency due to daisy-chaining and inefficiencies in link usage. The application input stream is bursty in nature and has more bank level locality than a random input stream. This results in significantly higher latencies for the applications than for the random input streams at the same system throughput.

5.1.2.1 Latency Characteristics

The variation of application read latency with the number of DIMMs in the channel and burst length are shown in figures 5.5 and 5.6 for FBD-DDR3 systems and figures 5.7 and for FBD-DDR2 systems. The read latency of the transaction is divided into queueing delay overhead and the transaction processing overhead. The queueing delay component refers to the duration the transaction waited in the queue for one or more resources to become available. Note that the overlap of queueing delay is monitored for all components except the memory controller request queue factor. The default latency cost is the cost associated with making a read request in an unloaded channel.

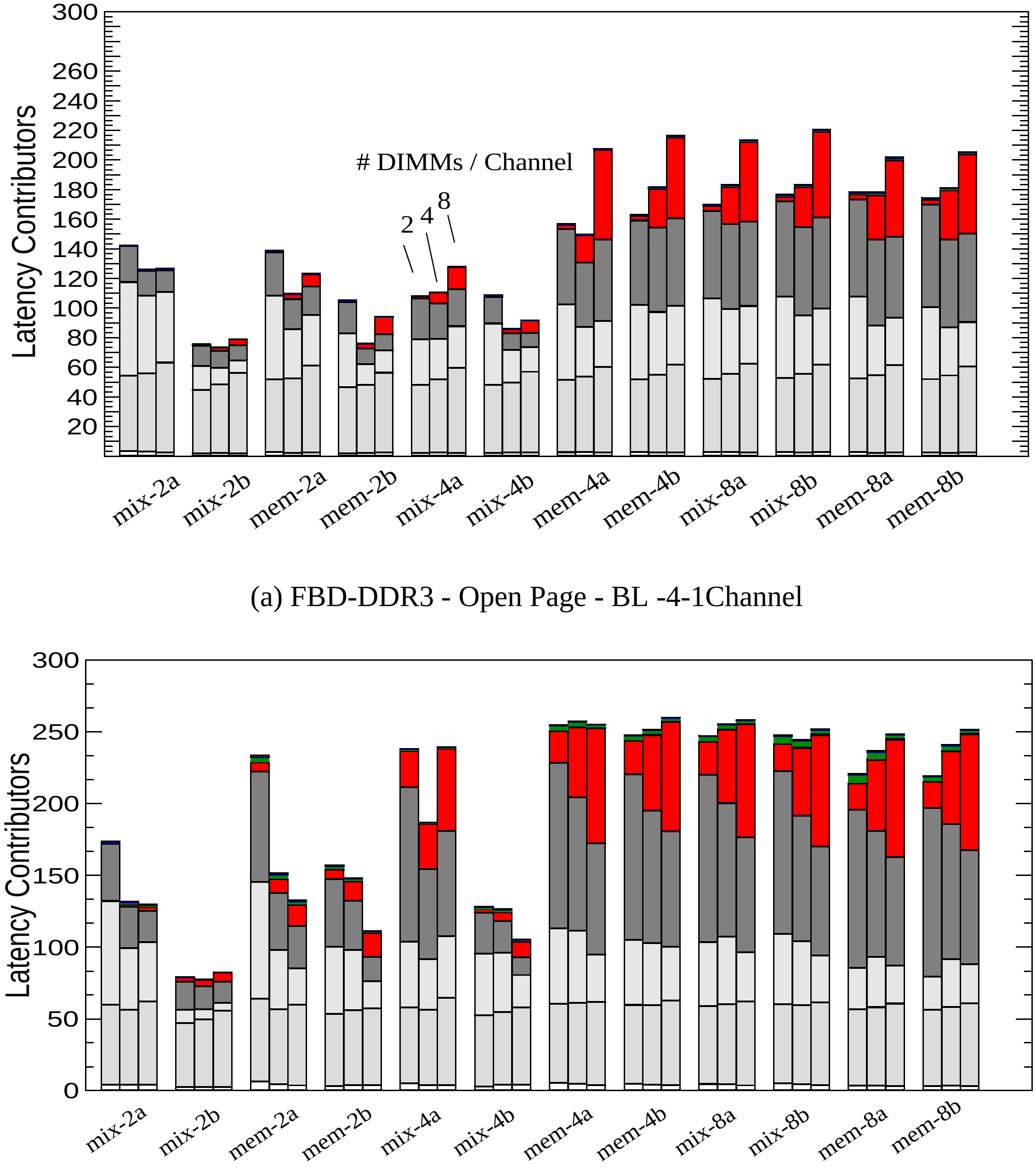
As seen earlier, increasing the channel depth results in a reduction in queueing delay due to DIMM unavailability, and a simultaneous increase in queueing delay due to the link being unavailable and the increased serialization cost. The queueing delay due to the north



(a) FBD-DDR3 - Closed Page - BL -4-1Channel

(b) FBD-DDR3 - Closed Page - BL -8-1 Channel

Figure 5.5. Latency contributors in a closed page FBD-DDR3 system. The figure shows the average read latency and its sources for the various applications for different channel depths. For a given application, the latency is shown in a set of bars, each bar corresponding to a different channel depth, going from 2 DIMM to 8 DIMM deep channels.

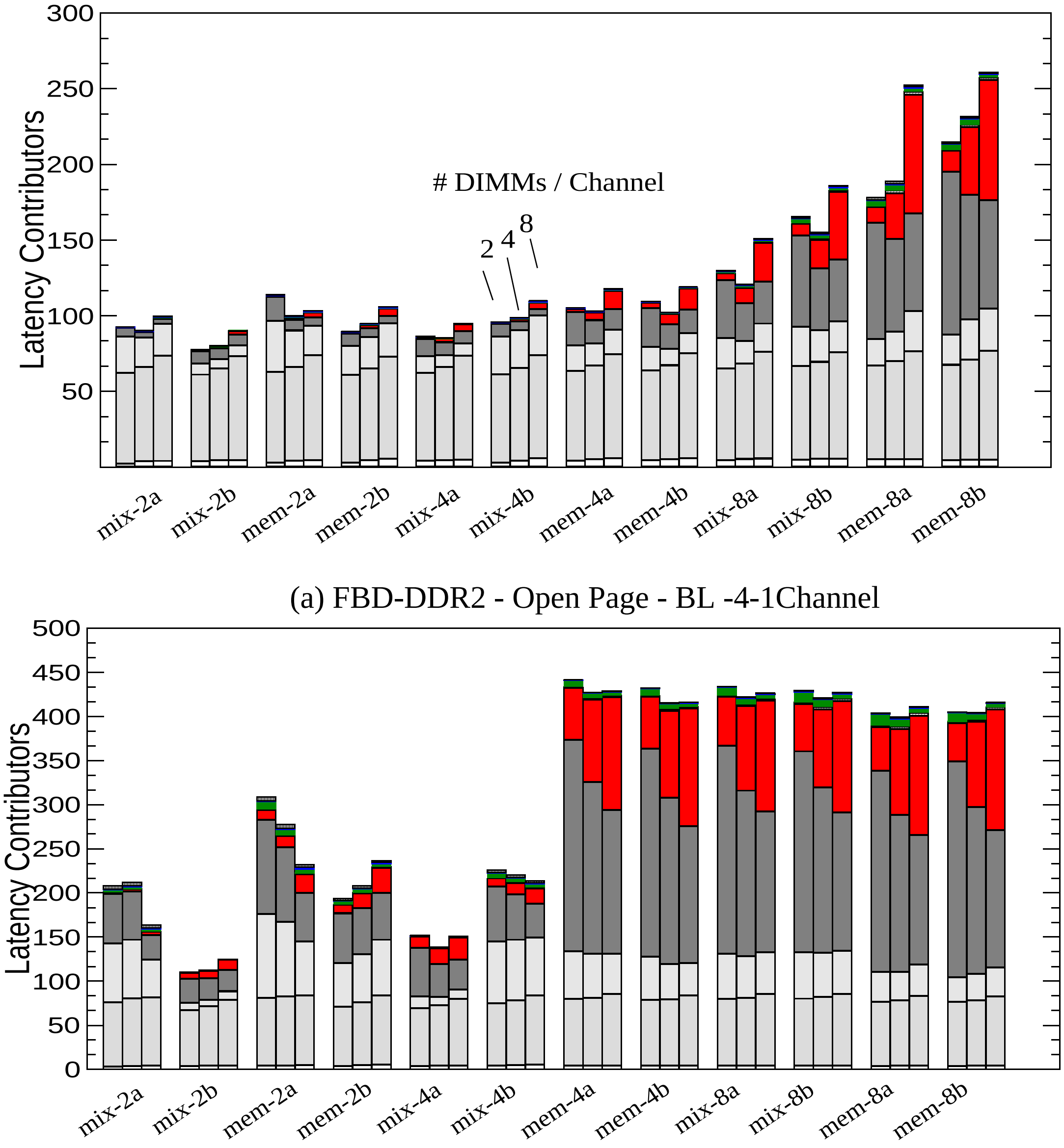


(a) FBD-DDR3 - Open Page - BL -4-1Channel

(b) FBD-DDR3 - Open Page - BL -8-1 Channel

Figure 5.6. Latency Contributors for Open Page System FBD-DDR3. The figure shows the average read latency and its sources for the various applications for different channel depths. For a given application, the latency is shown in a set of bars, each bar corresponding to a different channel depth, going from 2 DIMM to 8 DIMM deep channels.

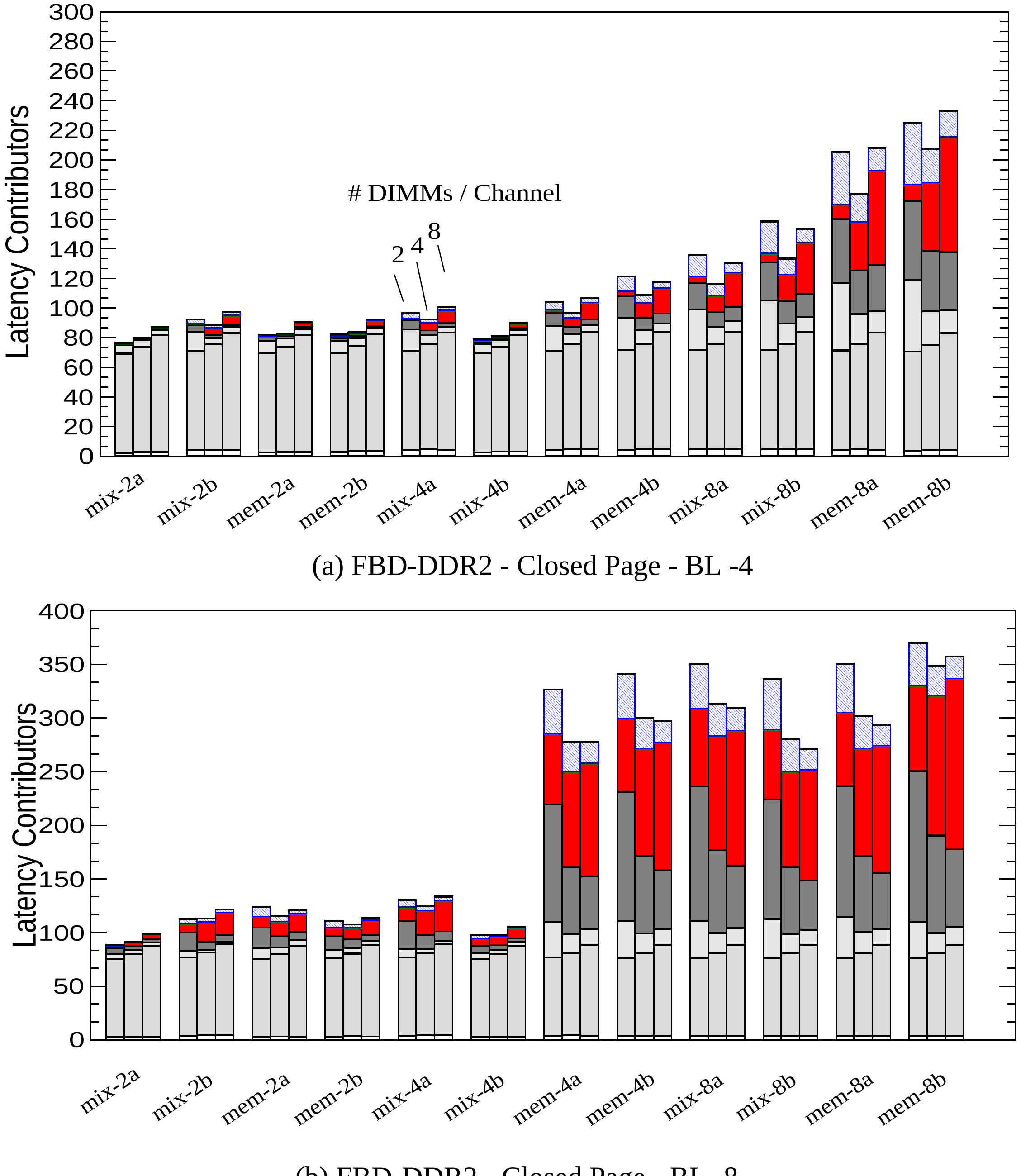
- ▩ South Link and DIMM
- ▨ South Link
- 🟩 South Link, DIMM, North Link
- ▨ South Link, North Link
- 🟥 North Link
- ■ DIMM, North Link
- □ DIMM
- □ Default Latency
- □ Transaction Queue



(a) FBD-DDR2 - Open Page - BL -4-1Channel

(b) FBD-DDR2 - Open Page - BL -8-1Channel

Figure 5.7. Latency Contributors for Open Page Systems. The figure shows the average read latency and its sources for the various applications for different channel depths. For a given application, the latency is shown in a set of bars, each bar corresponding to a different channel depth, going from 2 DIMM to 8 DIMM deep channels.



(a) FBD-DDR2 - Closed Page - BL -4

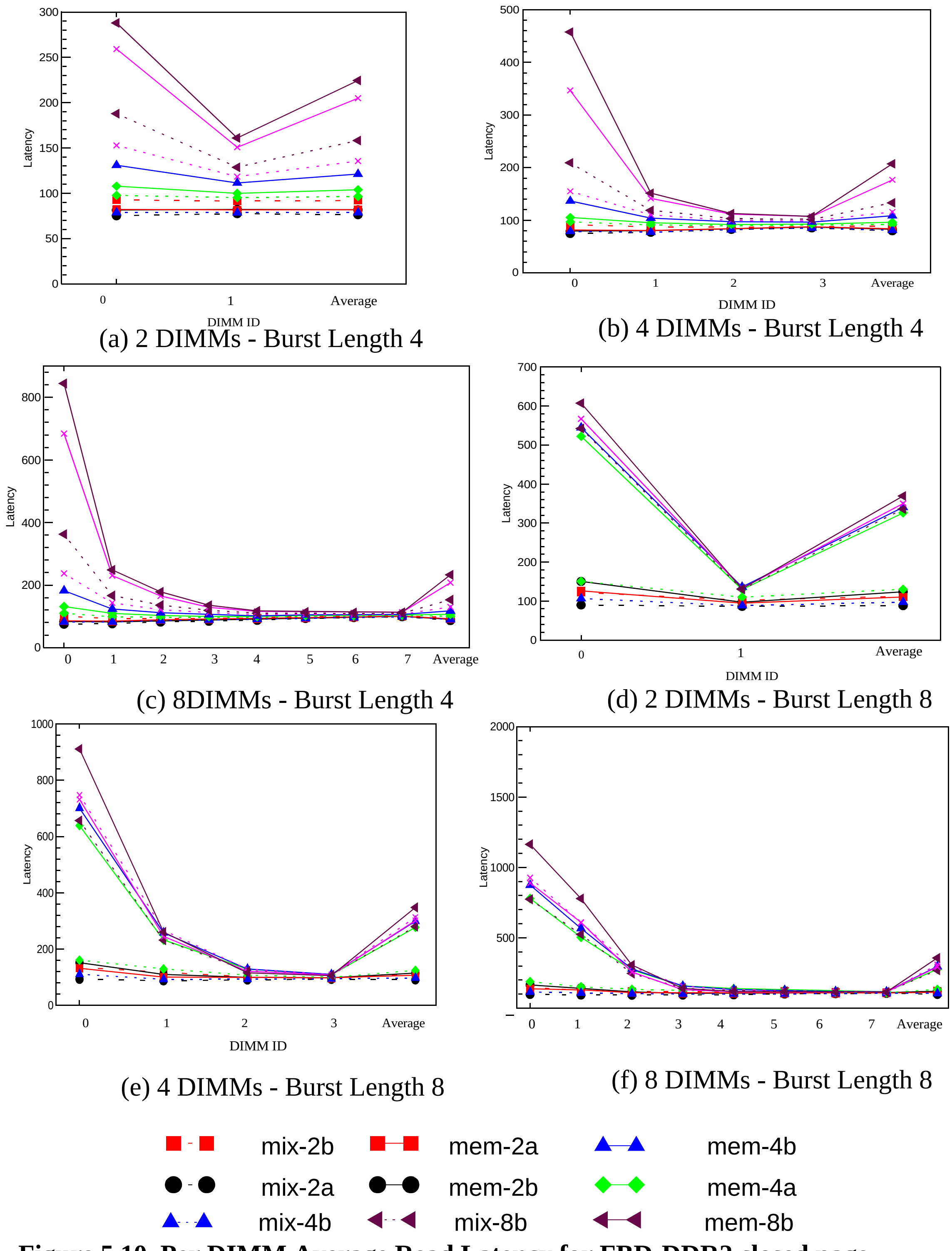
(b) FBD-DDR2 - Closed Page - BL -8

Figure 5.8. Latency Contributors for FBD-DDR2 Closed Page Systems. The figure shows the average read latency and its sources for the various applications for different channel depths. For a given application, the latency is shown in a set of bars, each bar corresponding to a different channel depth, going from 2 DIMM to 8 DIMM deep channels.

link being unavailable becomes larger in the a 8-DIMM deep channel. In an open page system, DIMM unavailability is a significant factor even for longer channels. Queueing delay due to the north link being unavailable contributes to under 10% to over a third of the observed read latency. The applications which experience the largest amounts of queueing delay are those with the largest bandwidth requirements. Burst length 8 systems experience larger queueing delays than burst length 4 systems because of the larger data transfers. For both burst length 4 and burst length 8 systems we look at identical cacheline sizes of 64 bytes. A single cacheline fill is satisfied in burst length 4 systems by 2 physical FBDIMM channels and in burst length 8 systems by one physical FBDIMM channel. In the former case the two physical FBDIMM channels are operated in lock-step.

5.1.2.2 Per-DIMM Latency Characteristics

Unexpectedly, read transactions that are addressed to the DIMM closest to the memory controller experience significantly higher read latency than transactions to other DIMMs in the system. This latency behavior was more prominent in systems with more DIMMs per channel and longer burst lengths. Figures 5.9, 5.10, 5.11 and 5.12, show the average read latency for a transaction to a particular DIMM in the system for FBD-DDR3 and FBD-DDR2 systems in both open page and closed page configurations. Note that open page configurations use the OBF scheduling policy while the closed page configurations use the RIFF scheduling policy and posted CAS. The DIMMs are numbered in increasing order starting with the DIMM adjacent to the memory controller. In addition to the average read latency for a given DIMM, the overall average read latency for the system is plotted as well.



(a) 2 DIMMs - Burst Length 4

(b) 4 DIMMs - Burst Length 4

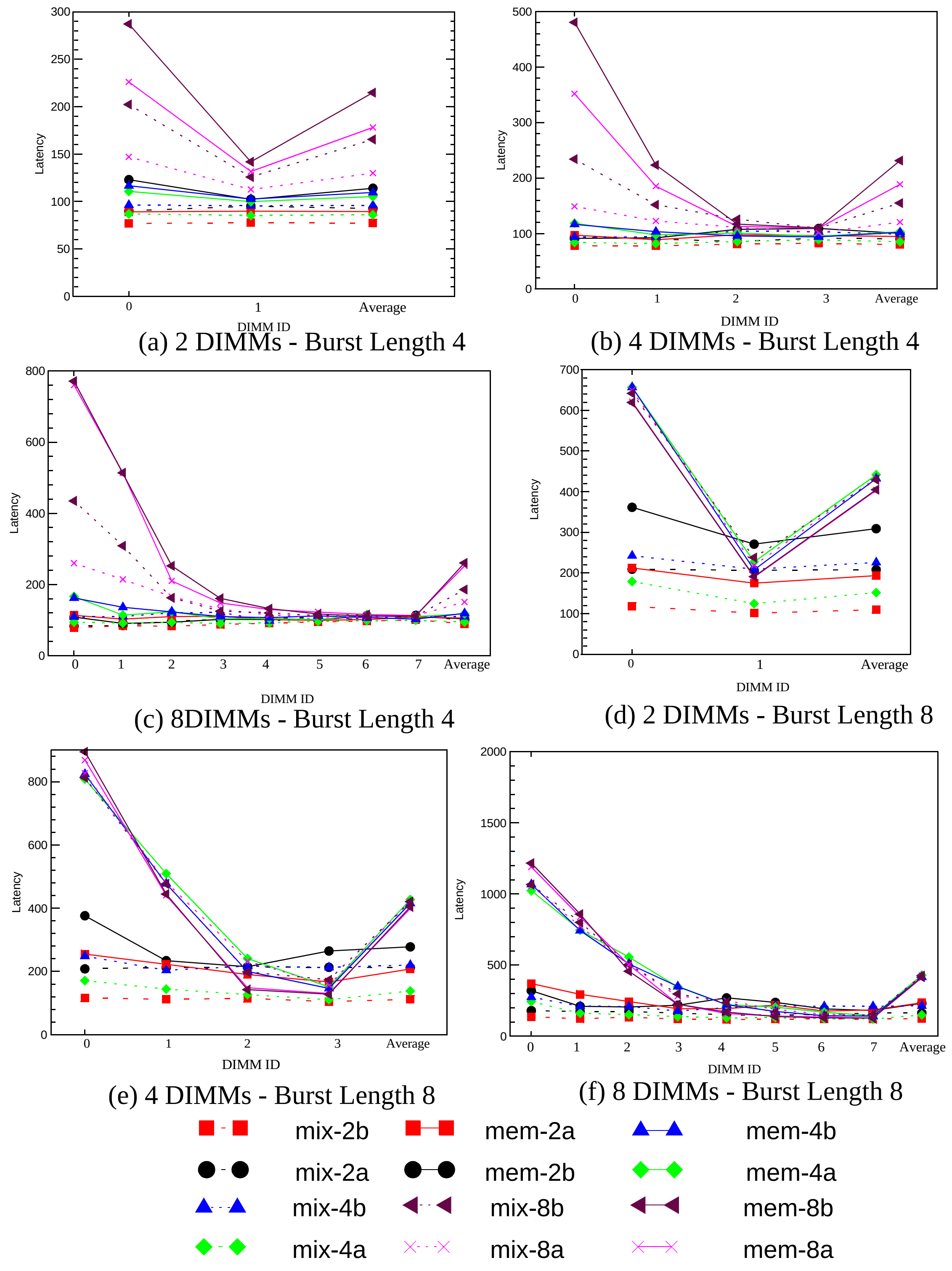
(c) 8DIMMs - Burst Length 4

(d) 2 DIMMs - Burst Length 8

(e) 4 DIMMs - Burst Length 8

(f) 8 DIMMs - Burst Length 8

Figure 5.10. Per DIMM Average Read Latency for FBD-DDR2 closed page systems. The graphs show the average read latency for a transaction addressed to a particular DIMM in the system. The DIMMs are ordered such that the DIMM closest to the memory controller has the lowest number. The last point on all the graphs is the overall average read latency



(a) 2 DIMMs - Burst Length 4

(b) 4 DIMMs - Burst Length 4

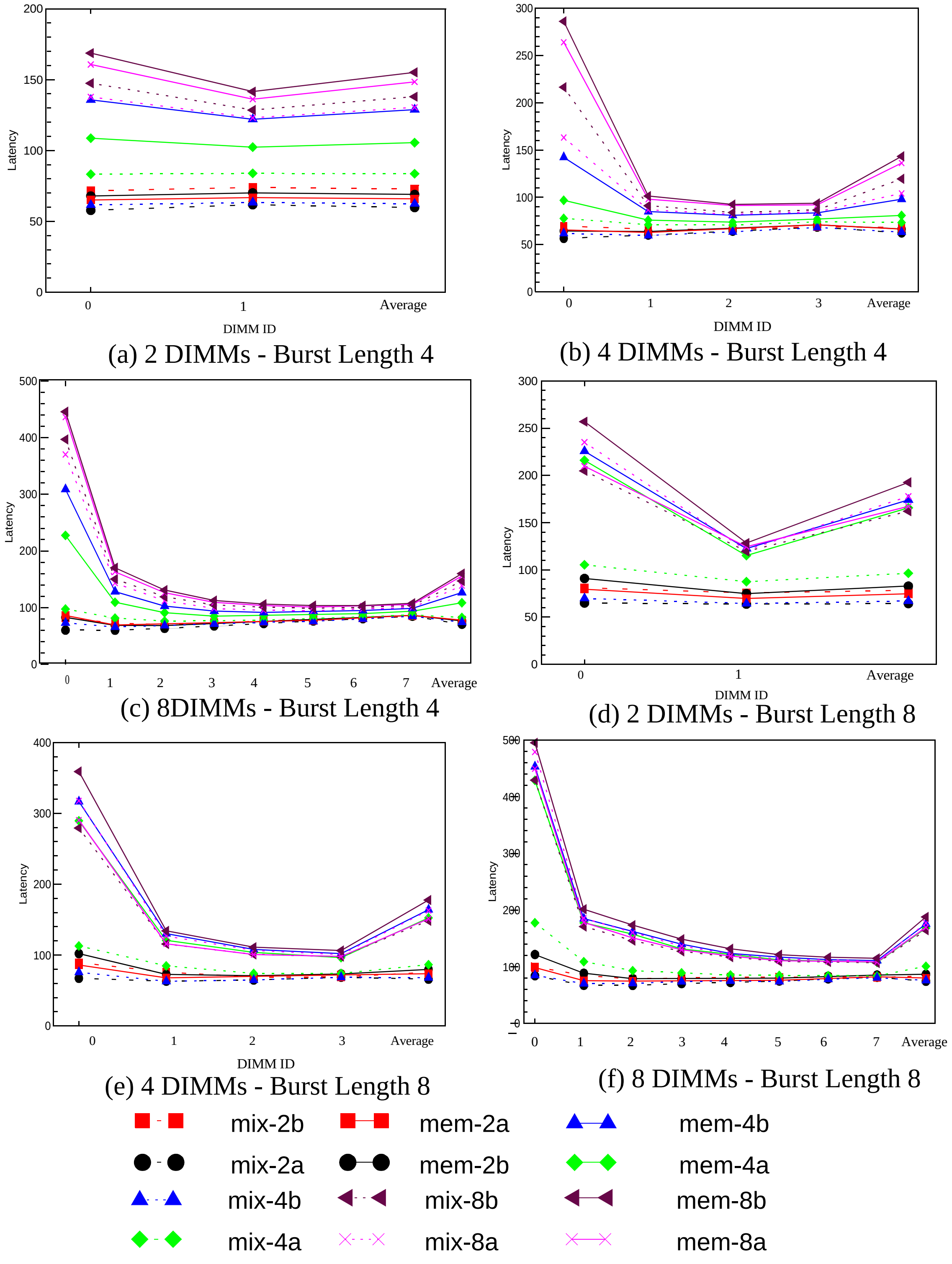
(c) 8DIMMs - Burst Length 4

(d) 2 DIMMs - Burst Length 8

(e) 4 DIMMs - Burst Length 8

(f) 8 DIMMs - Burst Length 8

Figure 5.11. Per DIMM Average Read Latency for FBD-DDR2 open page systems. The graphs show the average read latency for a transaction addressed to a particular DIMM/DIMM in the system. The DIMMs are ordered such that the DIMM closest to the memory controller has the lowest number. The last point on all the graphs is the overall average read latency.



(a) 2 DIMMs - Burst Length 4

(b) 4 DIMMs - Burst Length 4

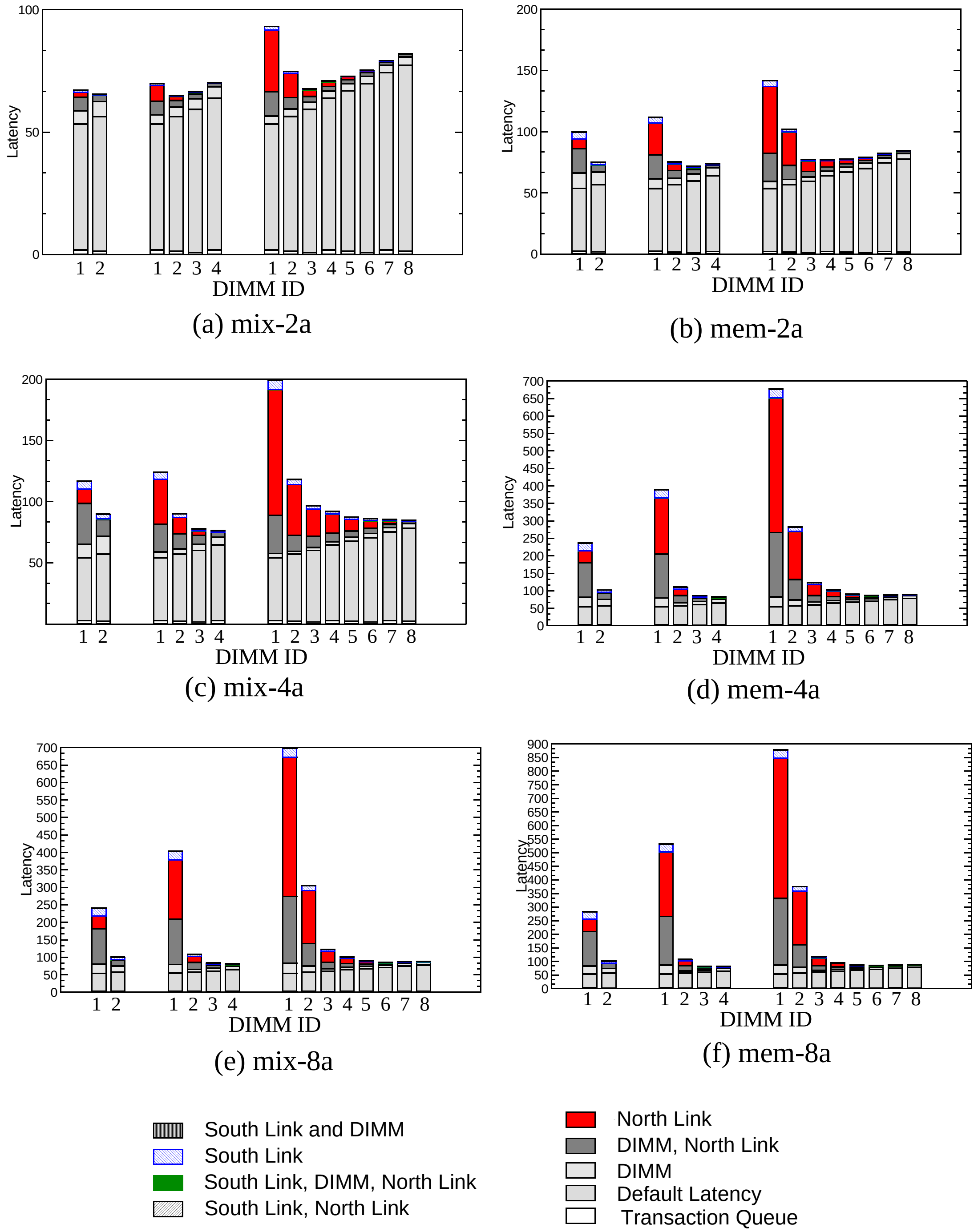
(c) 8DIMMs - Burst Length 4

(d) 2 DIMMs - Burst Length 8

(e) 4 DIMMs - Burst Length 8

(f) 8 DIMMs - Burst Length 8

Figure 5.12. Per DIMM Average Read Latency for FBD-DDR3 closed page systems. The graphs show the average read latency for a transaction addressed to a particular DIMM in the system. The DIMMs are ordered such that the DIMM closest to the memory controller has the lowest number. The last point on all the graphs is the overall average read latency



(a) mix-2a

(b) mem-2a

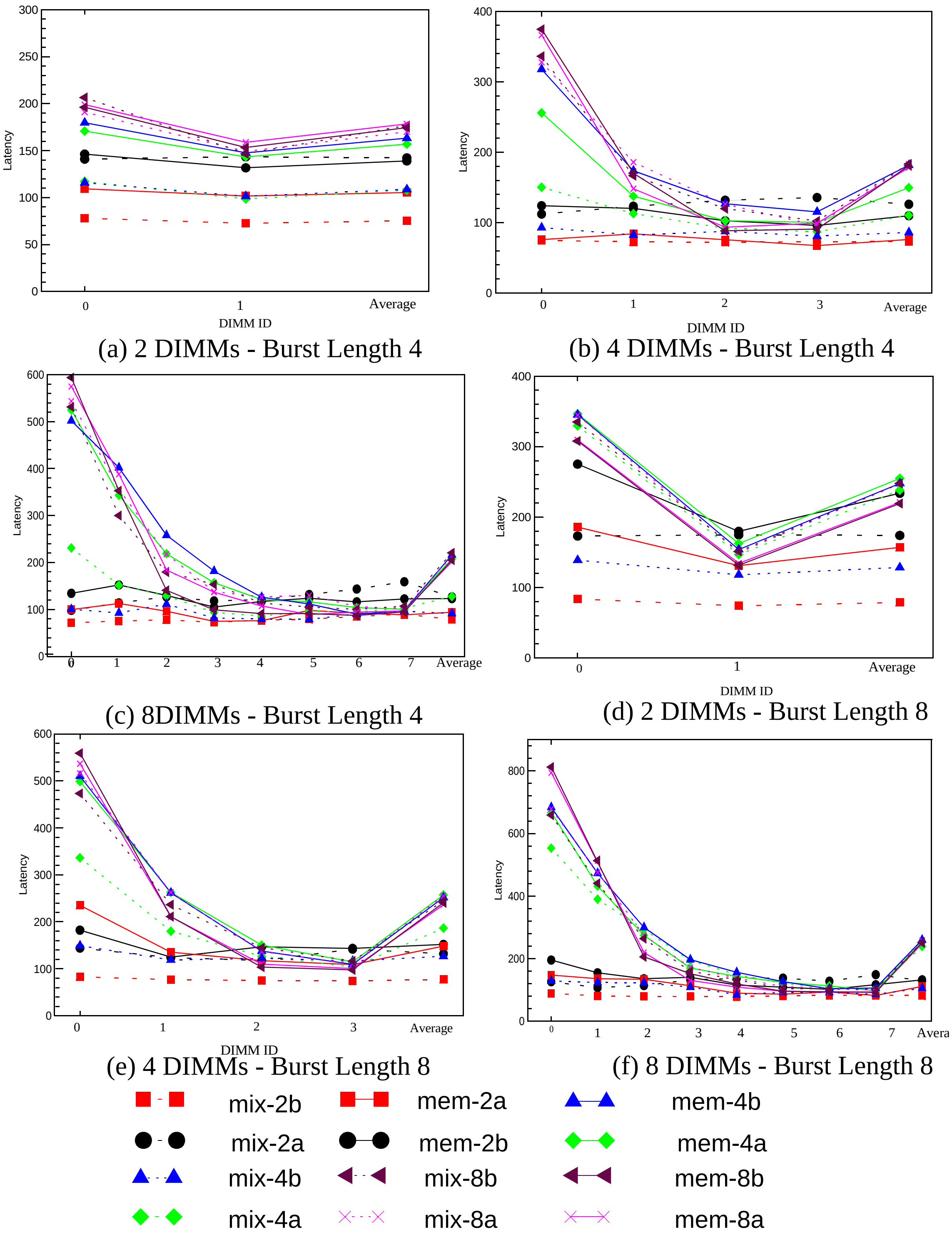
(c) mix-4a

(d) mem-4a

(e) mix-8a

(f) mem-8a

Figure 5.13. Per DIMM Average Read Latency Contributors for FBD-DDR3 closed page systems with burst length 8. The graphs show the contributors to average read latency for a transaction addressed to a particular DIMM in the system for different applications. The DIMMs are ordered such that the DIMM closest to the memory controller has the lowest number. The configurations are grouped on the x-axis such that the left-most group his a channel with 2 DIMMs while the right-most has



(a) 2 DIMMs - Burst Length 4

(b) 4 DIMMs - Burst Length 4

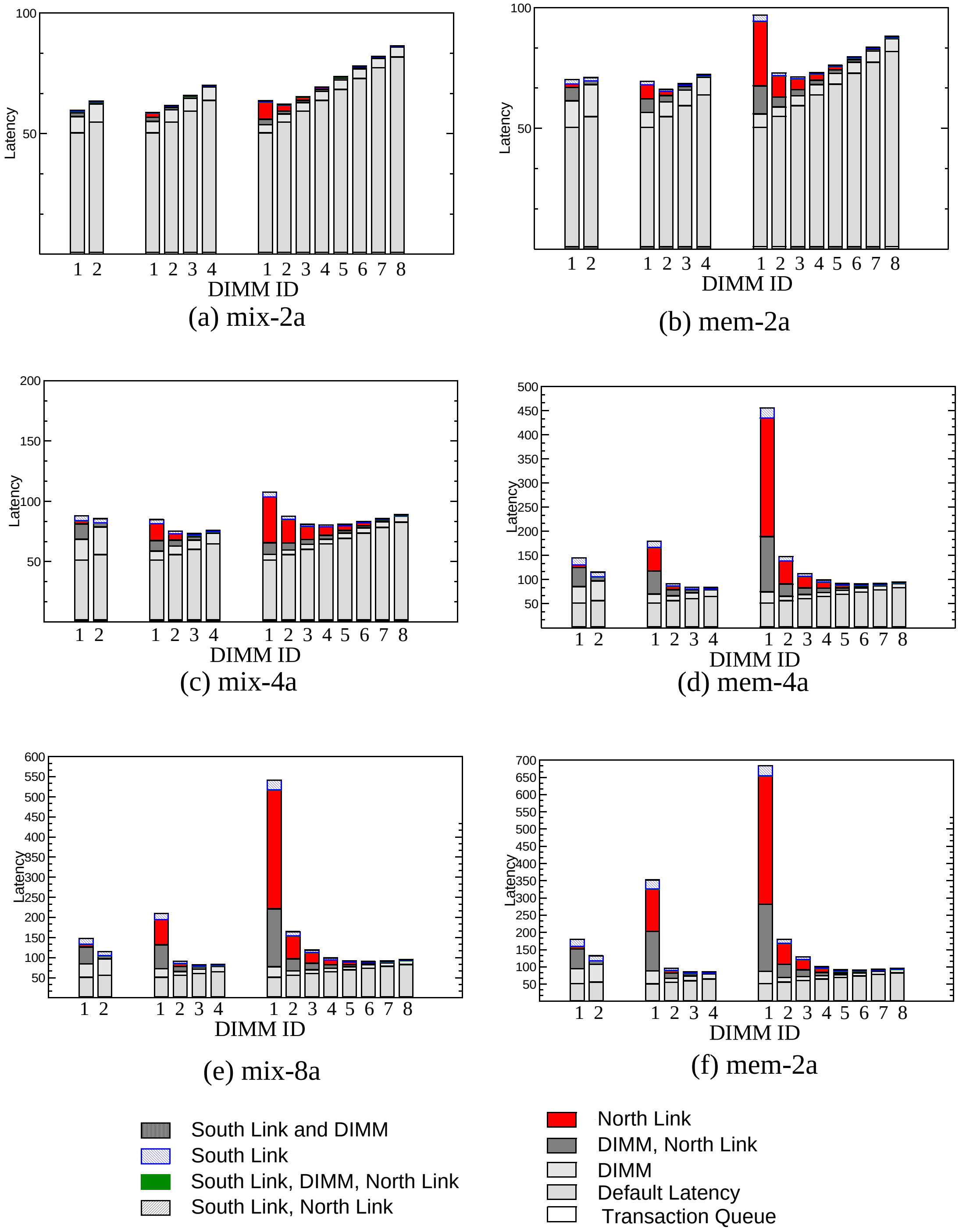
(c) 8DIMMs - Burst Length 4

(d) 2 DIMMs - Burst Length 8

(e) 4 DIMMs - Burst Length 8

(f) 8 DIMMs - Burst Length 8

Figure 5.9. Per DIMM Average Read Latency for FBD-DDR3 open page systems. The graphs show the average read latency for a transaction addressed to a particular DIMM in the system. The DIMMs are ordered such that the DIMM closest to the memory controller has the lowest number. The last point on all the graphs is the overall average read latency



(a) mix-2a

(b) mem-2a

(c) mix-4a

(d) mem-4a

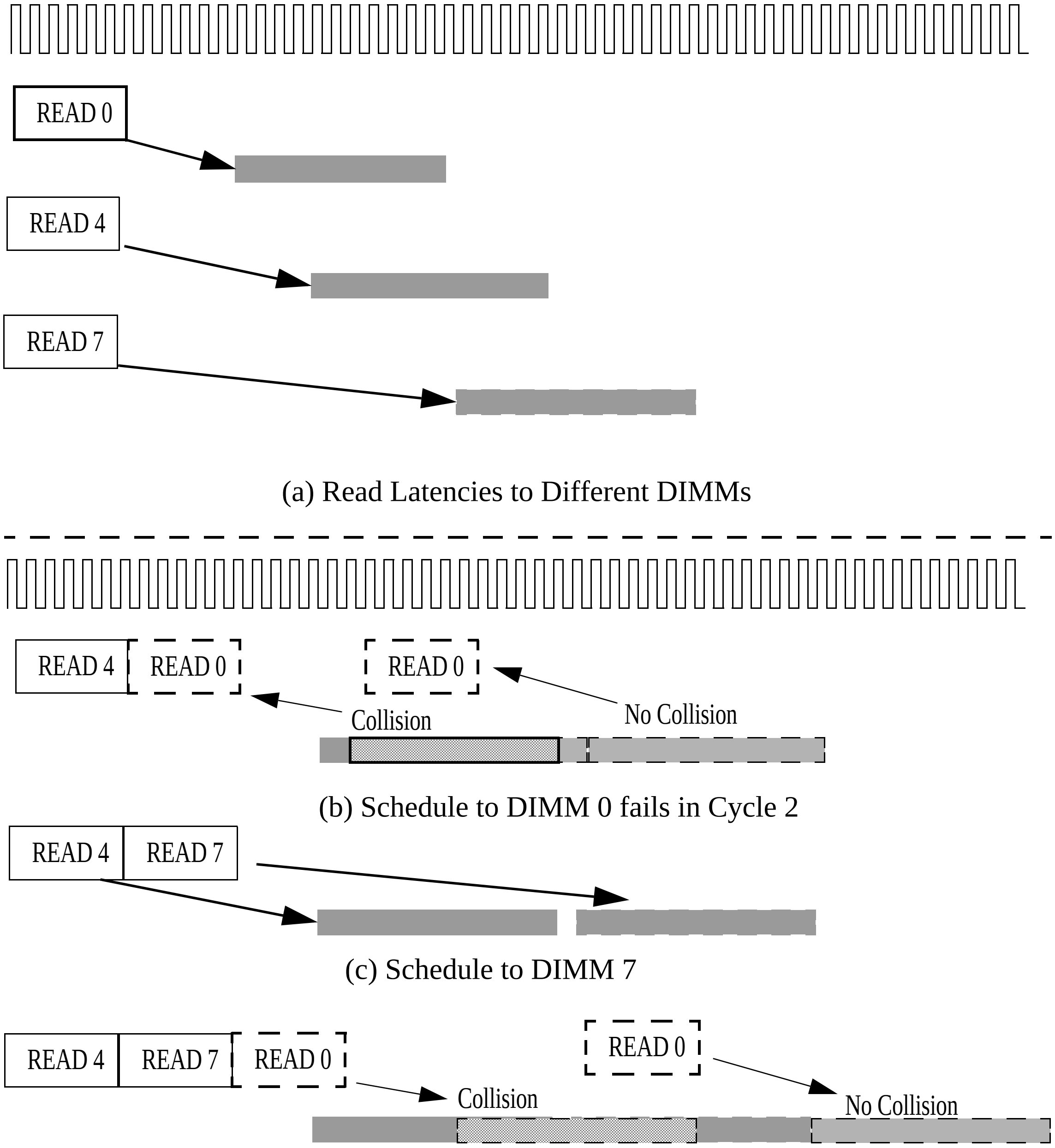
(e) mix-8a

(f) mem-2a

Figure 5.14. Per DIMM Average Read Latency Contributors for FBD-DDR3 closed page systems with burst length 4. The graphs show the contributors to average read latency for a transaction addressed to a particular DIMM in the system for different applications. The ranks are ordered such that the DIMM closest to the memory controller has the lowest number. The configurations are grouped on the x-axis such that the leftmost group his a channel with 2 ranks while the rightmost has 8 ranks.

For nearly all applications, the read latency experienced by a transaction to the first DIMM in the chain is higher than that experienced by a transaction to further DIMMs in the chain. This is very unexpected because in a variable latency mode system, the default round-trip cost for a read request to the first DIMM in the chain is lower than the default round-trip latency cost for a read request to any other DIMM in the chain.

The memory controller's scheduler uses a greedy approach to schedule return data on the north link. This approach inherently ends up being biased towards scheduling transactions that are scheduled to DIMMs further from the memory controller than those that are closer to the memory controller. To use a given north link slot, the memory controller schedules a read request to a DIMM further in the chain earlier than it would have to if that read request was to a DIMM closer to the controller. Hence, it is more likely that the memory controller at a given scheduling instant will schedule requests to DIMMs further from the memory controller. Figure 5.15 illustrates the phenonmenon that due to the different latencies on the buses, a read to DIMM 0 has to be issued later than a read to DIMM 7. By pushing the point at which the memory controller will issue a transaction to a closer DIMM further in time, a greedy memory controller will starve requests to the first DIMM in the chain. This delay in scheduling requests tapers off as the distance of the DIMM from the memory controller increases. Hence, although the highest latency cost is associated with the last DIMM in the chain, the first DIMM in the chain has the worst read latency. Figures 5.13 and 5.14 shows that the read latency for transactions to the first DIMM is dominated by north link unavailability and this queueing delay increases with increase in the number of DIMMs in the channel.



(a) Read Latencies to Different DIMMs

(b) Schedule to DIMM 0 fails in Cycle 2

(c) Schedule to DIMM 7

(d) Schedule to DIMM 0 fails in Cycle 3

Figure 5.15. Why Scheduling to closer DIMMs fails? The top part of the figure shows the read latencices for read requests to different DIMMs in the same channel, numbered in increasing order as you move away from the memory controller. Part (b) depicts the memory controller having to make a decision as to what to schedule in the cycle immediately after the read to DIMM 4. Figure b shows that a read to DIMM0 cannot be scheduled till 2 command cycles later without their being a collision on the bus. Figure d shows that the request to DIMM 7 can be issued in cycle 2. As expected, in cycle 3, again the read to DIMM 0 cannot be issued. However, due to the previously scheduled read to DIMM 7, the read to DIMM 0 can be scheduled only in command cycle 5.

However, in the case of the mixed workloads with 2-4 threads and for the memory intensive workload with 2 threads in some configurations the read latency of transactions to the first DIMM in the chain is not the highest. In these systems, queueing delay due to the north link being unavailable does not dominate because of the lower bandwidth utilization of these applications. In these cases we see that the read latency increases with the distance of the DIMM from the memory controller.

5.2. Re-ordering of Data Returns

In this section we describe the implementation details for memory controller structures needed to permit re-ordering of data returns on the return channel. The FBDIMM protocol is a deterministic protocol where the memory controller is aware of when the operation initiated by a transmitted command will complete, when the data burst will begin at the DIMM and when it should arrive at the memory controller. In short, the memory controller is aware of all system state and is by specification and design required to ensure that all system timing requirements are met. Traditionally, memory controller design has dealt with read requests with identical latency values and therefore guaranteed to return in the same order as the commands were sent out. Re-ordering return data has never been required.

The different round-trip latencies for a read in the FBDIMM channel makes it possible to have some reads return prior to previously scheduled reads in the system. This would require the memory controller not just to be able to identify when to schedule reads but also be able to associate the returning read data with the appropriate transaction. In this section we first describe techniques that the memory controller can use to identify north link violabilities and finally modifications to the memory controller queue to enable it to associate the correct read request with the returned data.

5.2.1 Mechanisms to track North Link Usage

Current memory controller design requires the memory controller to keep track of only the last use of the northbound link. To allow read data frames to be transmitted during idle times between two read transmissions, the memory controller has to be able to track multiple link uses. In the following sections, we describe two mechanisms that can be used to track north link usage. The first approach we looked at was a status table based mechanism that tracks link usage on a cycle by cycle basis and is used to identify the upper bounds on benefits achieved by re-ordering data returns. The second approach uses *n*-counters to keep track of the last *n* uses of the north link.

5.2.1.1 Status Table Mechanism

In this scheme the busy duration of the last link in the northbound channel is monitored per channel cycle. This is done using a circular buffer with a head and a tail. The head points to the slot which corresponds to the time at which a transaction scheduled to the northernmost DIMM would require the northernmost link to transmit data. The tail points to the entry which corresponds to the time at which a read transaction to the southernmost DIMM in the system would complete data transmission.

The exact time offset for the head and tail are

Head $t_{BUS} + t_{DIMM}$

Tail $2Rt_{BUS} + t_{DIMM} + NUM\_FRAMES * t_{FRAME},$

where, t_{BUS} is the link transmission overhead.

T_{DIMM} is the overhead for processing the transaction on the DIMM - this includes the overhead to decode the command packet, to retrieve data from the DRAM and to pack the read data into frames.

T_{FRAME} is the duration for a single frame to be transmitted.

NUM_FRAMES is the total number of frames required to burst all the read data back to the memory controller. *NUM_FRAMES* is a function of the burst length and is in general burst length/2.

t_{LINK} is the duration of a single cycle on the link. *R* is the total number of DIMMs in the channel.

Each slot monitors the link use for a single channel clock cycle. The status table size can be given as

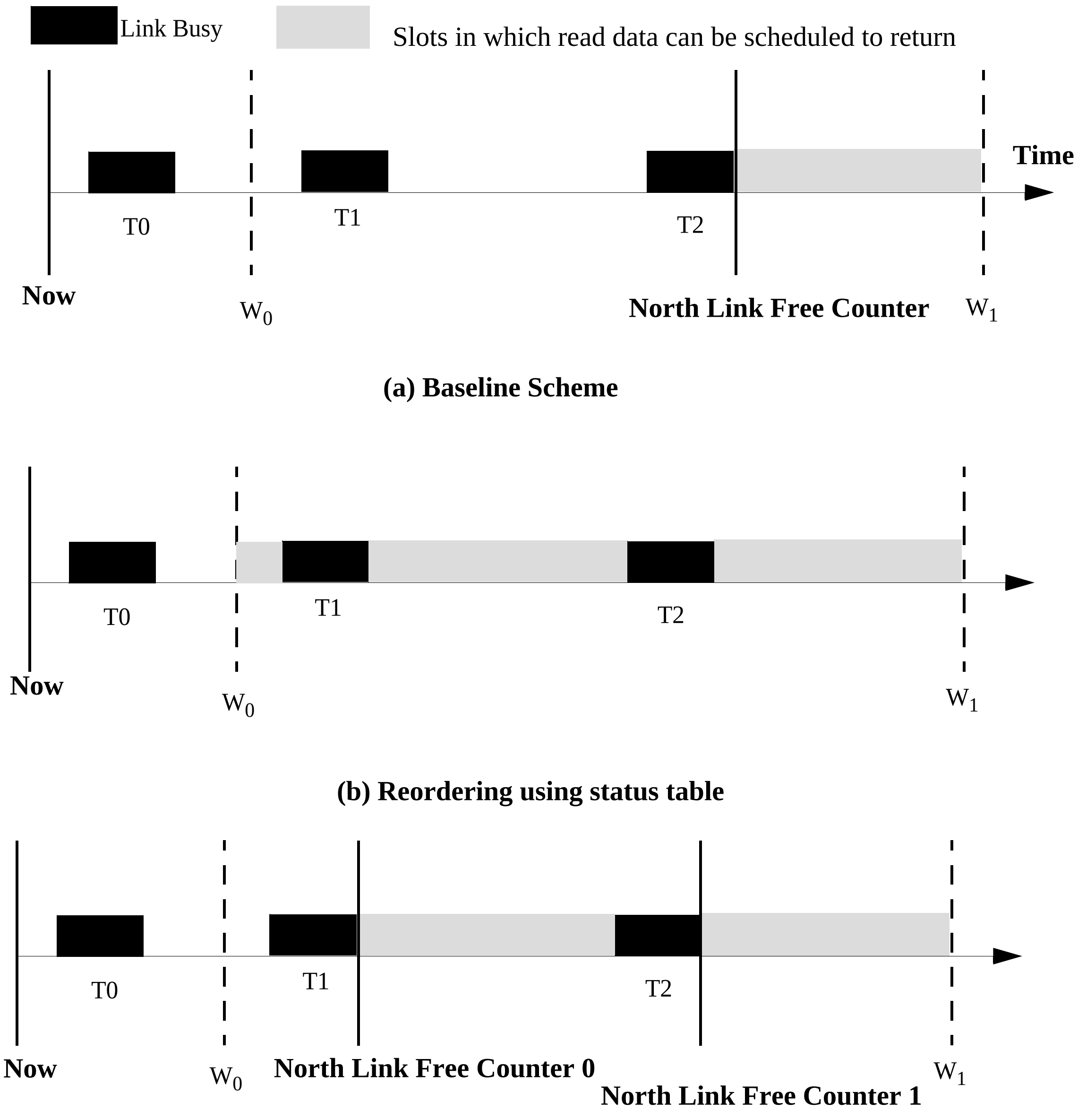
Buffer Size = (Tail Time Offset - Head Time Offset) / Frame Duration.

Buffer Size $= (2*R*t_{BUS} + t_{DIMM} + NUM\_FRAMES * t_{FRAME} - (t_{BUS} + t_{DIMM\ })/ t_{FRAME}$

Buffer Size $= NUM\_FRAMES + (2R -1)*t_{BUS}/ t_{LINK}$

5.2.1.2 Counter-based Mechanism

Fig 5.16 depicts the usage of the northernmost link in the system from the current scheduling instant, represented as *Now*, to W_1, the last instant at which a transaction that is scheduled *Now* can use the link. Filled black slots are used to identify periods when the link is utilized by returning read data. Free slots are marked by either blank space or by grey slots. The window of time when transactions that are scheduled *Now* may want to use the link is marked by the window W_0 to W_1. W_0 is the earliest that a read transaction will require this link, while W_1 is the latest that a transaction to the last DIMM will require the link.


Link Busy
Slots in which read data can be scheduled to return
Time
T0
T1
T2
Now
W_0
North Link Free Counter
W_1
T0
T1
T2
Now
W_0
W_1
T0
T1
T2
Now
W_0
North Link Free Counter 0
North Link Free Counter 1
W_1

(a) Baseline Scheme

(b) Reordering using status table

(c) Reordering using Counters

Figure 5.16. Monitoring North Link Availability in Scheduling Windows. The figure shows the points at which different re-ordering schemes can schedule a transaction to return. Each figure depicts the busy and free duration of the northern most link using a time-line starting from now i.e. the point at which the transaction is being scheduled, to W_1, the last instant that a transaction scheduled now will use the north link. The filled grey slots represent busy durations, where read data for a previously scheduled transaction uses the link. The dotted grey slots represent the points at which the scheme in question can schedule a read transaction to return. W_0 represents the earliest point at which a transaction may require the north link. W1 represents the latest point at which the north link will be required. North Link Free Counter are the counters used by the two counter based schemes to keep track of end of a link busy point.

The baseline scheme which does not permit re-ordering uses a single counter to keep track of the last use of the southernmost link in the system, shown in the Fig 5.16as North Link Free Counter. As seen from Fig 5.16 (a), the scheduler can schedule only a transaction that returns data only after the last read burst has completed. The status table based scheme described earlier is shown in Fig 5.16 (c). This scheme is able to schedule a transaction that can return at any point in the entire window - W_0 to W_1, provided the link is free.

The counter-based mechanism is a simple scheme that extends the baseline scheme so that it can increase the window that the scheduler may schedule a read transaction to return data in. This scheme uses two counters instead of one to keep track of the busy duration of the north link. The counters keep track of the latest two uses of the north link or the last two points at which data is returned. By using two counters, the memory controller is able to now schedule transactions to return read data in the idle time between the end of the last two known uses of the bus in addition to the duration after the completion of the last transaction. Fig 5.16 (b) shows that the counter based scheme increases the window within which a read data can be scheduled to be returned.

Using two counters allows us to schedule read data to return in two slots. Similarly using *n* counters would enable us to be able to schedule transactions to return in *n* slots. The maximum number of counters required thus would be a function of the maximum number of read data bursts which could be fitted into a single link scheduling window. In general, the number of counters required would be one less than the total number of read transactions that can be scheduled in a given window. The upper bound on the number of transactions that can fit into a scheduling window is given by the following equation,

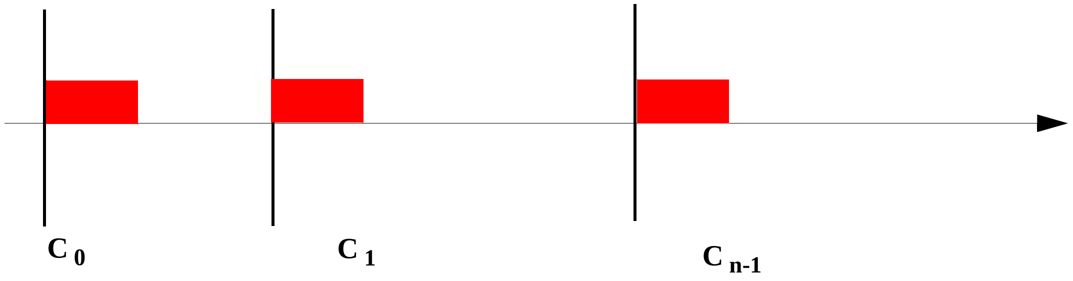


Figure 5.17. Counter Mechanism to keep track of North Link Busy Periods. C_n to C_0 are used to keep track of the start of periods of link utilization as shown in the figure.

Upper Bound on Read Transactions = Window Size / Duration of Single Read Operation

$= 2Rt_{BUS} + t_{DIMM} + NUM\_FRAMES * t_{FRAME} - t_{BUS} -t_{DIMM}/NUM\_FRAMES * t_{FRAME}$

$= ((2R - 1)t_{BUS}+ NUM\_FRAMES * t_{FRAME}/NUM\_FRAMES * t_{FRAME}$

As seen in the equation above, the number of read transactions which can be used to fill the window is a function of the transmission overhead and the number of frames required for a single read data burst.

N-Counter Implementation. In this section we describe how to implement a scalable *n*-counter solution to monitor north link utilization. Each of the *n* counters, C_0 to C_{n-l}, keeps track of the start of the last *n* busy durations of the link. Fig 5.17 illustrates how the busy durations are tracked by a counter. The counters are sorted so that, C_0 keeps track of earliest use of the north link, while C_{n-1} keeps track of the last use of the north link.

When the memory controller makes a scheduling decision it has to ensure that there is no collision on the data bus. The transaction is schedulable in an idle gap between data bursts represented by counters C_i and C_{i+1}, the following condition is satisfied,

$$(L_{START} > C_i + t_{READDATA})(L_{END} < C_{i+1})$$

L_{START} is the time at which the link transmission starts,

L_{END} is the end of the link transmission time, and

$t_{ReadData}$ is the duration the read data takes to be transmitted back.

Since the link usage prior to that captured by C_0 is not known, the following condition also has to be guaranteed,

$$C_0 < L_{START}$$

After making a scheduling decision, the counters have to be updated with the link completion time. The mechanism to implement this counter update is similar to the priority queue implementations used in high-speed packet switch architectures[4]. Fig 5.18 shows the implementation of a shift register based counter update mechanism.

The shift register queue has as many nodes as there are counters. In the figure, the shift register queue is drawn so that by moving to the left in the chain one encounters counters that monitors earlier bus use. Each node has as input the counter value and a shift signal from the node on its right. Besides these inputs, a node also has an update signal, a

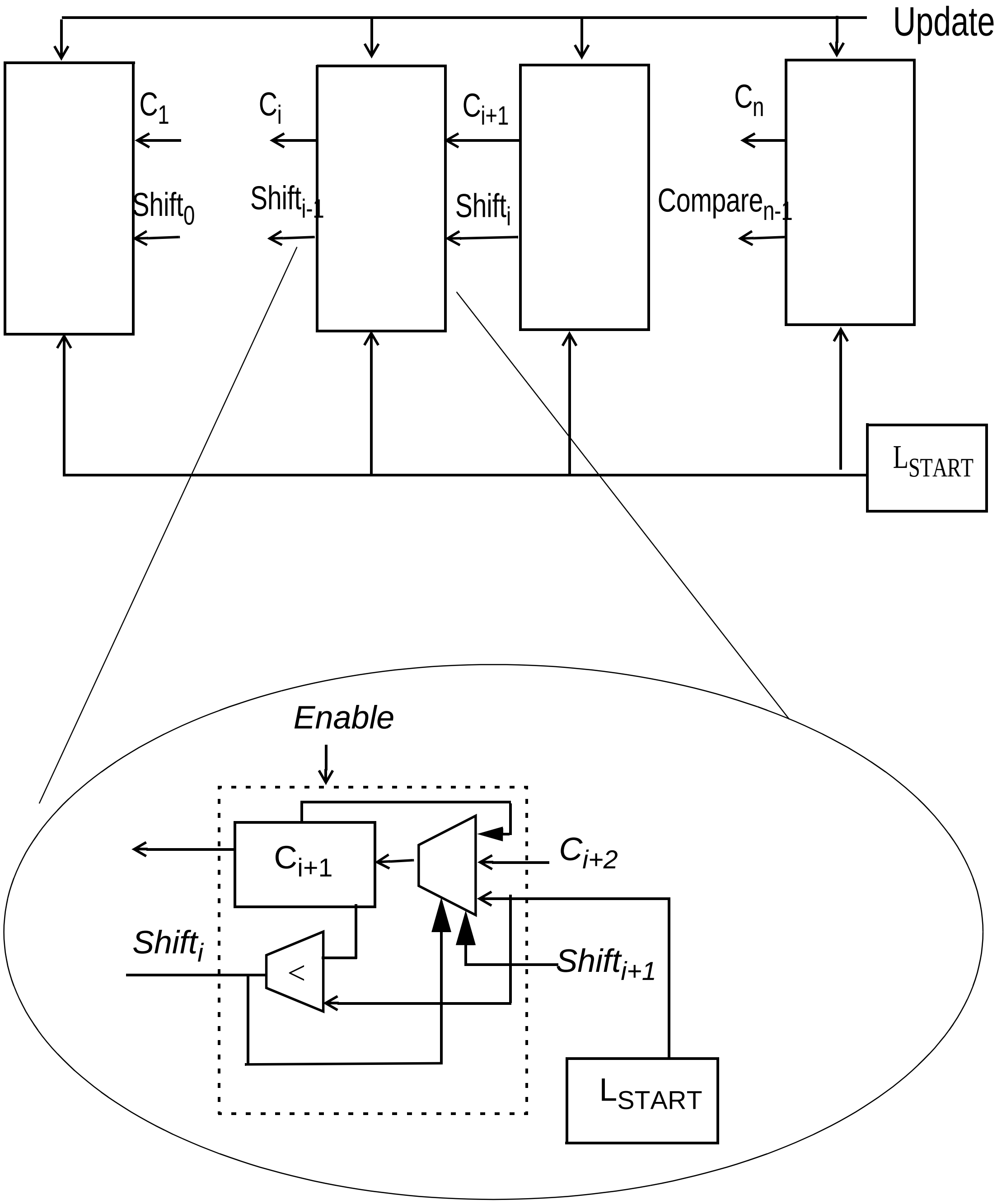


Figure 5.18. Counter Update Mechanism. The figure shows a shift-register architecture used to update the counter values. Signal $Shift_i$ is used to determine if the counter value is replaced by the adjacent counter value or by the new link update time. Each node has a built in comparator which is used to generate the shift signals.

read value signal (not shown in the figure) and the newly scheduled transaction link transmission start time being input into it. Each node comprises of a mux which determines whether to overwrite the current counter value with the input counter value or by the new link usage time or to retain the current value. This is dependent on the relation of the current counter value and its immediate right neighbor node's counter value with the new link usage time.

The logic defining how C_i should be updated is given by the following logical relations.

$C_i = C_{i+1}$, when $Shift_i = C_{i+1} < L_{START}$,

$C_i = L_{START}$, when $C_i < L_{START}\ \&!Shift_i$

$C_i = C_i$, when $C_i > L_{START}\ \&!Shift_i$

Since each node is modular the *n*-counter shift register queue can be easily scaled to arbitrarily large sizes. Unfortunately,the scalability in implementation does not extend to the performance of the n-counter shift register because of the necessity of broadcasting last link utilization time to each node.This should not be an issue at lower DRAM bus speeds where the maximum number of counters required, as seen in the previous section, is two or three.

The logic to determine whether to update the node's counter value can be integrated into the scheduling logic which determines whether the transaction's data burst experiences any conflicts on the north link. In this case the logic in each node can be simplified further.

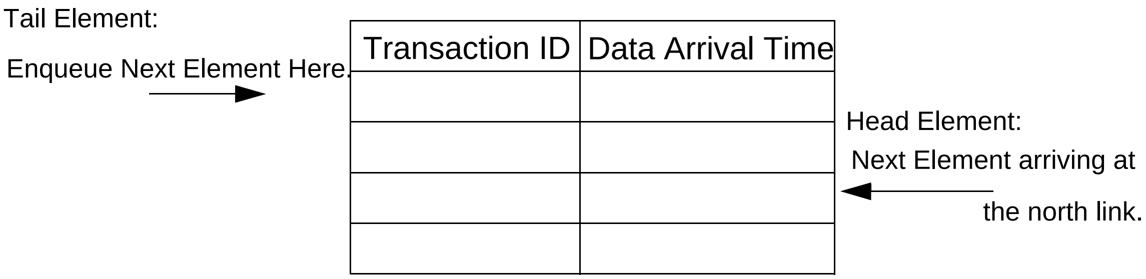


Figure 5.19. FIFO Response Queue. The response queue holds the transaction ID and data arrival time. The latter is used to determine when the data is available at the link and by the memory controller to track data arrival.

5.2.2 Mechanism to re-order the Response Queue

Traditional memory controller design uses a FIFO read response queue which is shown in Fig 5.19. The response queue holds the transaction ID or request ID used by the CPU or requesting component that is used to identify the read transaction making the request and in some cases the expected arrival time of the data. The returning read data is associated with the read transaction at the head of the queue. The read transaction ID is popped from the top of the queue and then sent with the returned read data to the processor. Memory controllers can use the data arrival time to identify when data is arriving at the memory controller. Arrival times are typically relative to the start of a data return of the previously scheduled transaction. To support the re-ordering of return data, the response queue should be re-ordered to reflect the new order in which data is returned.

5.2.2.1 Self Reordering Queue

The self-reordering queue is a read response queue which has been enhanced to allow reordering. Like the read response queue, each queue entry holds the returning order of the entry, the ID of the transaction it refers to, and a valid bit. The valid bit indicates the availability of each entry.

The returning order indicates when the transaction completes. Entries are addressed by the returning order and this order is updated dynamically. The dynamic returning order can be maintained in the form of self-increment/self-decrement counters, shift registers, etc. The transaction that is to complete first or at the head of the returning order is associated with the returning order zero. Each entry can be read when it reaches the head of returning order and can be read only once.

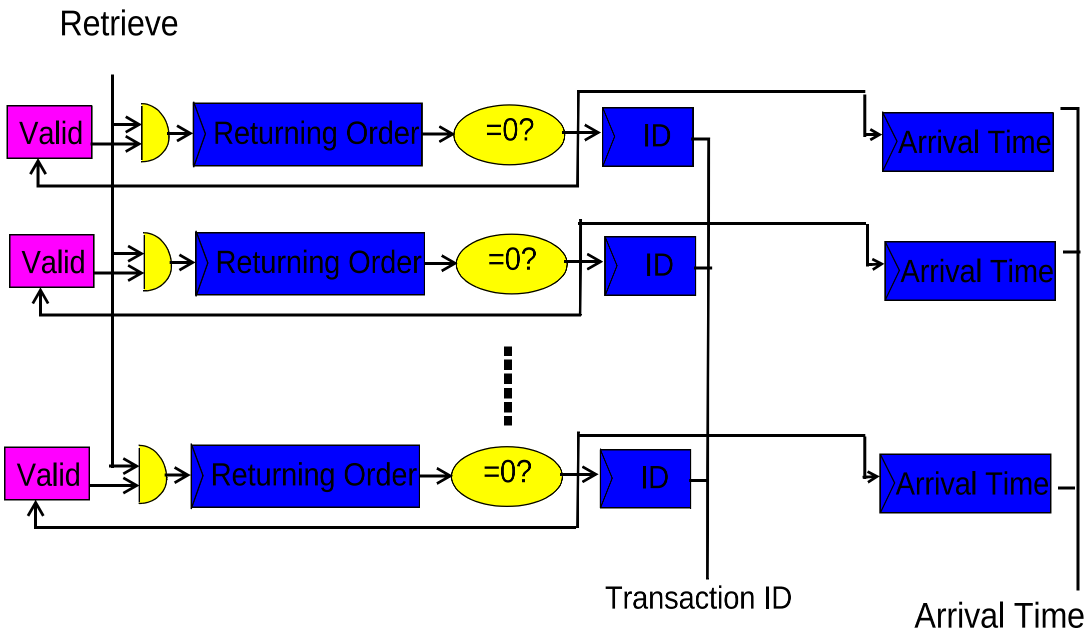
The main operations required to access the queue and update it upon the issue of a memory request include

- Retrieve
- Reorder

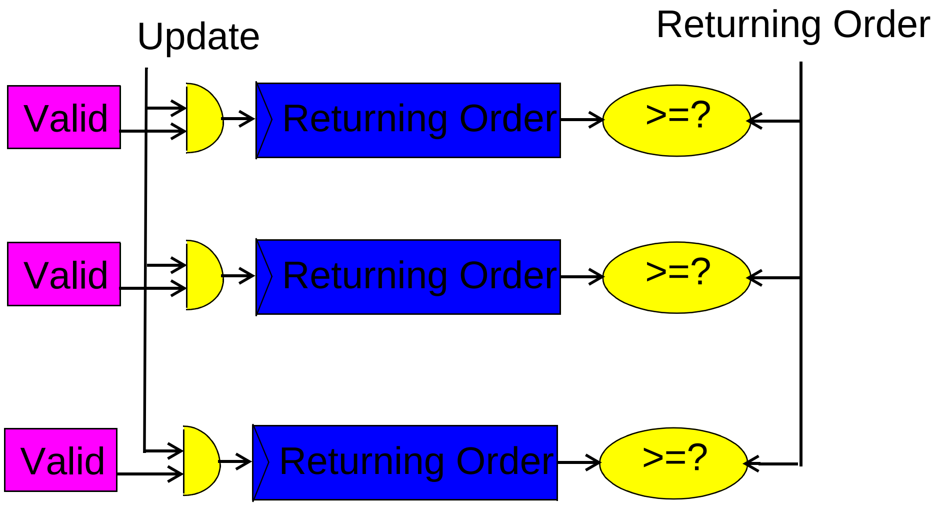
Retrieve Operation. Fig 5.20(a) shows how the retrieve operation is implemented in hardware. Retrieve is performed when read data arrives at the memory controller. The memory controller has to retrieve the transaction id associated with the read data and ship this back to the memory controller with the read data.

The retrieve signal is asserted and this initiates the comparison operation to determine which transaction ID has returning order zero. Upon a hit, the transaction id is read out and all the returning orders have to be updated. The arrival time is read out as well.

Reorder Operation. Fig 5.20 (b) shows the hardware implementation of the update logic. Reorder operations are performed when a read request is issued from the memory control-

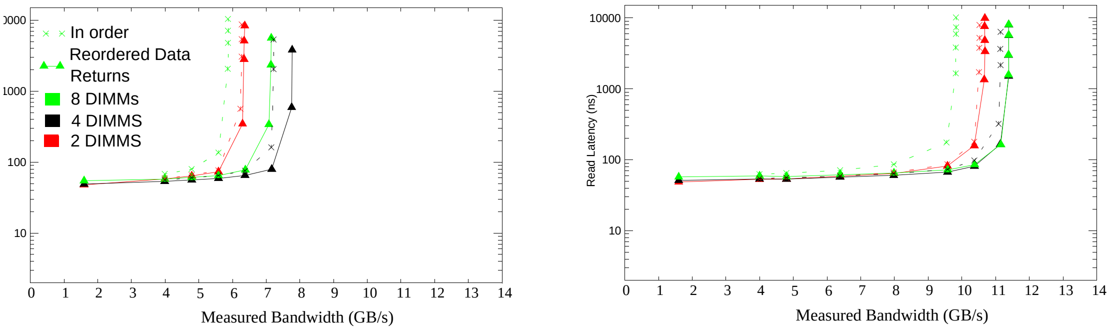


(a) Self-reordering Queue: Retrieve



(b) Self-reordering Queue: Update / Reorder

Figure 5.20. Self-reordering Queue Logic. The figure shows the operation of the self-reordering queue. The queue has to be updated on a retrieve and a reorder operation.



(a) Burst length 4

(b) Burst length 8

Figure 5.21. Impact of permitting out-of-order return on northbound link. The figures show the latency bandwidth characteristics for systems with and without reordering of data returns on the northbound link. The dotted lines are for the in-order return, while the solid lines are for a system that allows reordered data return.

ler to the memory. The processor-issued transaction ID needs to be stored in the self-reordering content-addressable memory with the appropriate entry ID. Further the returning order of the other read requests will have to be updated.

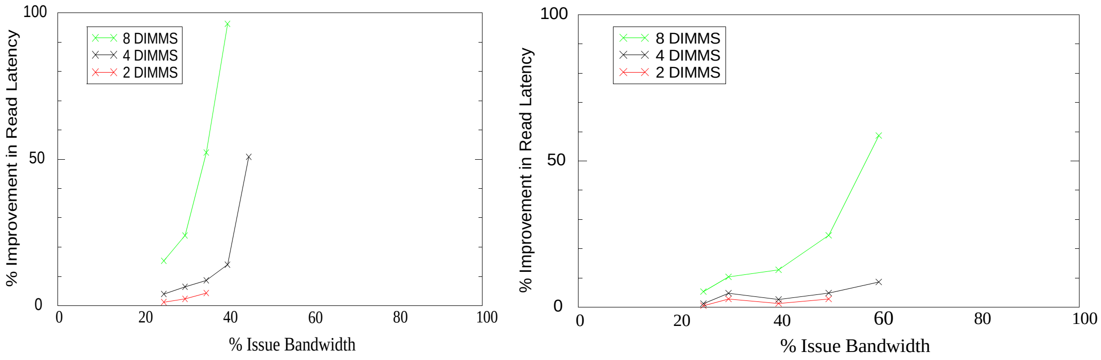
The main steps involved in this process are

- Determine the returning order of the issued request.
- Update the returning order of existing outstanding requests.
- Write its transaction ID and the corresponding returning order to an available entry.

5.2.3 Re-ordering Data Returns : Limit Study

In this section, we describe the latency and bandwidth improvements gained by permitting re-ordering of data returns. By using random address traces, we found that the maximum bandwidth of a 8-DIMM FBDIMM channel increased by nearly 1.5 Gbps.

Latency Bandwidth Characteristics. Permitting reordering of read data returns improves the average read latency by significantly reducing queueing delays associated with the



(a) Burst Length 4

(b) Burst Length 8

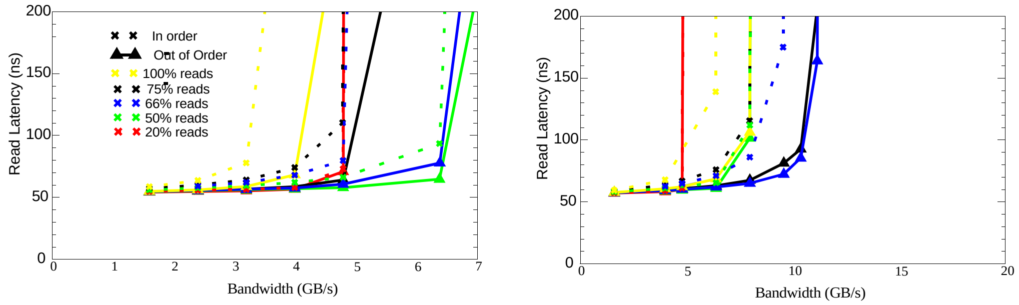
Figure 5.22. Improvements in Read Latency by permitting reordered data return of read data. The graphs show the improvements in read latency by permitting reordered data return of read data. Note that the improvements are only shown for points which are just prior to the knee of the latency-bandwidth curve. The x-axis is given in terms of percentage of bandwidth requested by the incoming address stream.

north link being unavailable. This reduction varies from 5-100% and contributes to an increase in the maximum bandwidth of nearly 1-2 GBps. The benefits are more apparent in systems with deeper channels.

In figures 5.21 and 5.22, the impact of allowing reordered data return of read data can be seen. Data for all graphs in this section unless specified are for the status table . Reordering read data return is most effective for systems with deeper channels. To effectively use this technique, the memory controller has to find an idle gap in north link use which is large enough to carry an entire read data burst. For a system with burst length 4, this is 2 read-data frame periods and for a burst length 8 system this is 4 read-data frame period.This makes the improvements from using this approach sensitive to the channel depth.

Fig 5.22 shows the improvements in read latency for different DIMM configurations for a system with 66% read traffic. Improvements in read latency are only shown for the operating region which is prior to the knee of the latency-bandwidth curve. Out of order return is effective in lowering read latency by 10-100%. The reductions in latency are significant when approaching the knee of the latency bandwidth curve. The efficacy of the scheme is a function of both the burst length and channel depth. Hence, we see that burst length 4 systems benefit from the scheme for channels with 4 or more DIMMs, while burst length 8 systems see benefits mainly in 8 DIMM deep channels.

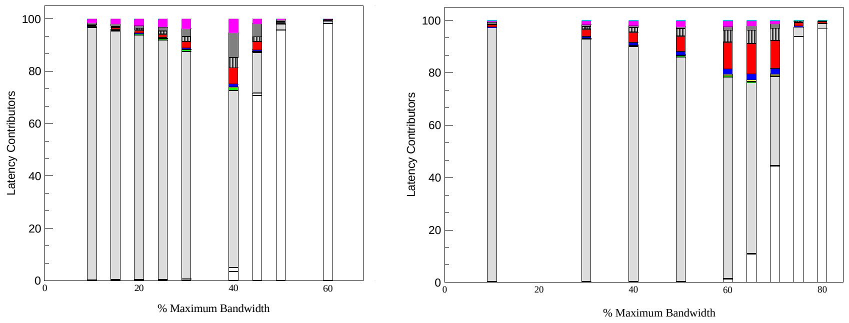
Fig 5.23 shows how input streams with different proportions of read traffic benefit from reordering of transactions on the return bus. As can be seen in the figure input streams with a small proportion of read traffic e.g. 20% in the graphs do not benefit from the out-of-order return. This is because these streams are constrained by the availability of south link bandwidth and not by the availability of north link bandwidth. As the proportion of read traffic is increased in the input stream, the use of out-of-order return improves. This is because the amount of north link utilization increases as well. Thus we see that for streams with 50-75% read traffic there is around a 1-2 GBps by doing away with in order return. At



(a) Burst Length 4/ #DIMMs 8

(b) Burst Length 8 / #DIMMs 8

Figure 5.23. Impact of permitting out-of-order return with different ratio of read traffic. The graphs show the improvements in read latency by permitting reordered data return of read data. Note that the improvements are only shown for points which are just prior to the knee of the latency-bandwidth curve. The x-axis is given in terms of percentage of bandwidth requested by the incoming address stream.



(a) #DIMMs 8, Burst length 4

(b) #DIMMs 8, Burst length 8

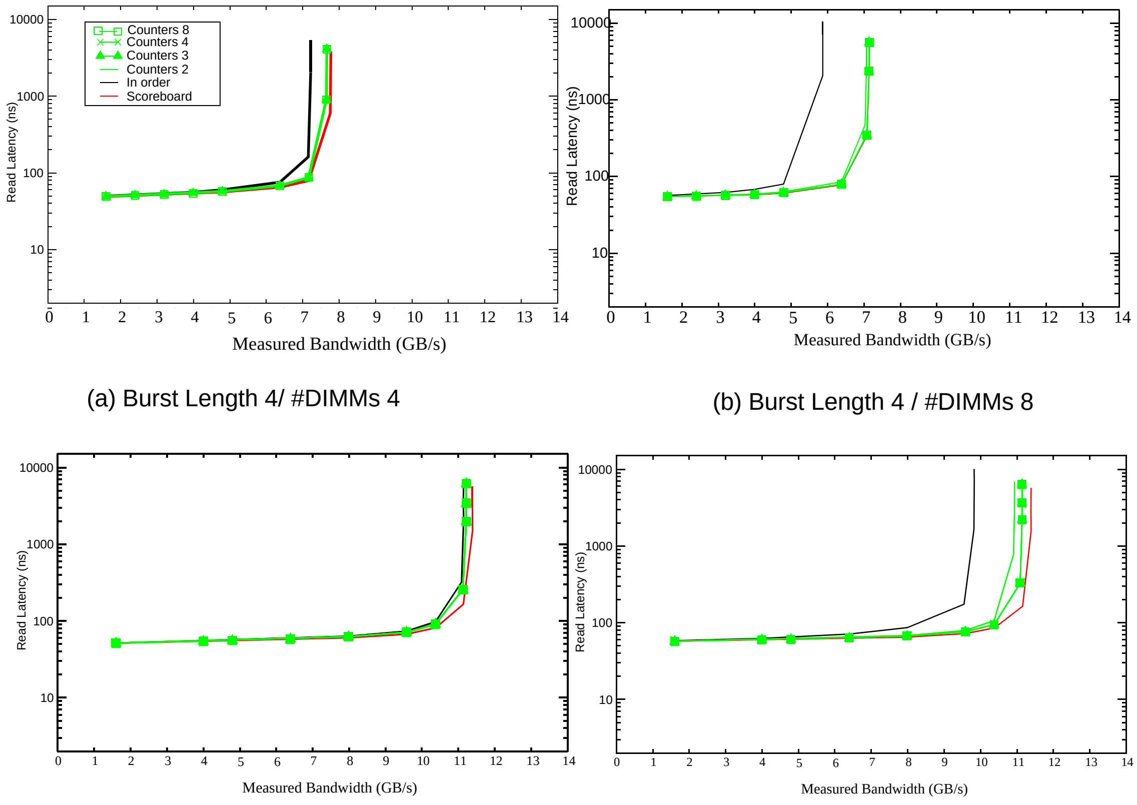
Figure 5.24. Latency contributors for system using reordered data return. the graphs show how the various components for systems which support out-of-order return using a status table . Note that the x-axis is given in terms of issue bandwidth and the y-axis is in terms of percentage. The latency value is normalized to a 100%.

higher read traffic proportions (100% in this case), the benefit are less, because the larger use of the north link. All data is shown for a 8 DIMM deep channel. As observed earlier, the benefits diminish when growing to smaller channel depths. Hence the graphs for systems with lesser number of channels is not shown.

Fig 5.24 shows what the chief contributors to latency are for a system using out-of-order return. By comparing figures 5.24 (a) and (b) with figures 5.24 (e) and (f) respectively, we can see that the queueing delay due to the north link being unavailable has been reduced but not eliminated.

Out of order methodologies. The performance improvement obtained by using a counter based scheme with 2 or more counters is comparable to that of the scheme using status table . The improvements in sustainable bandwidth by increasing the counters from 2 to 3 is marginal and nearly non-existent for further increases in the number of counters.

Fig 5.25 shows how using the counter based reordering scheme improves the latency bandwidth characteristics of a FBDIMM channel. The improvements by using a counter based reordering scheme are identical to those with using a status table for burst length 4 systems. In the case of burst length 8 systems the latency bandwidth curves are very close. The improvements in using additional counters is not apparent in any configuration and only marginally in the 8 DIMM deep system which uses a burst length of 8. Although the



(a) Burst Length 4/ #DIMMs 4

(b) Burst Length 4 / #DIMMs 8

(c) Burst Length 8/ #DIMMs 4

(d) Burst Length 8 / #DIMMs 8

Figure 5.25. Impact of different reordering schemes on Latency bandwidth characteristics. The figures show how the latency - bandwidth characteristics of the system for the counter-based reordering schemes compare with the baseline system and the status table.

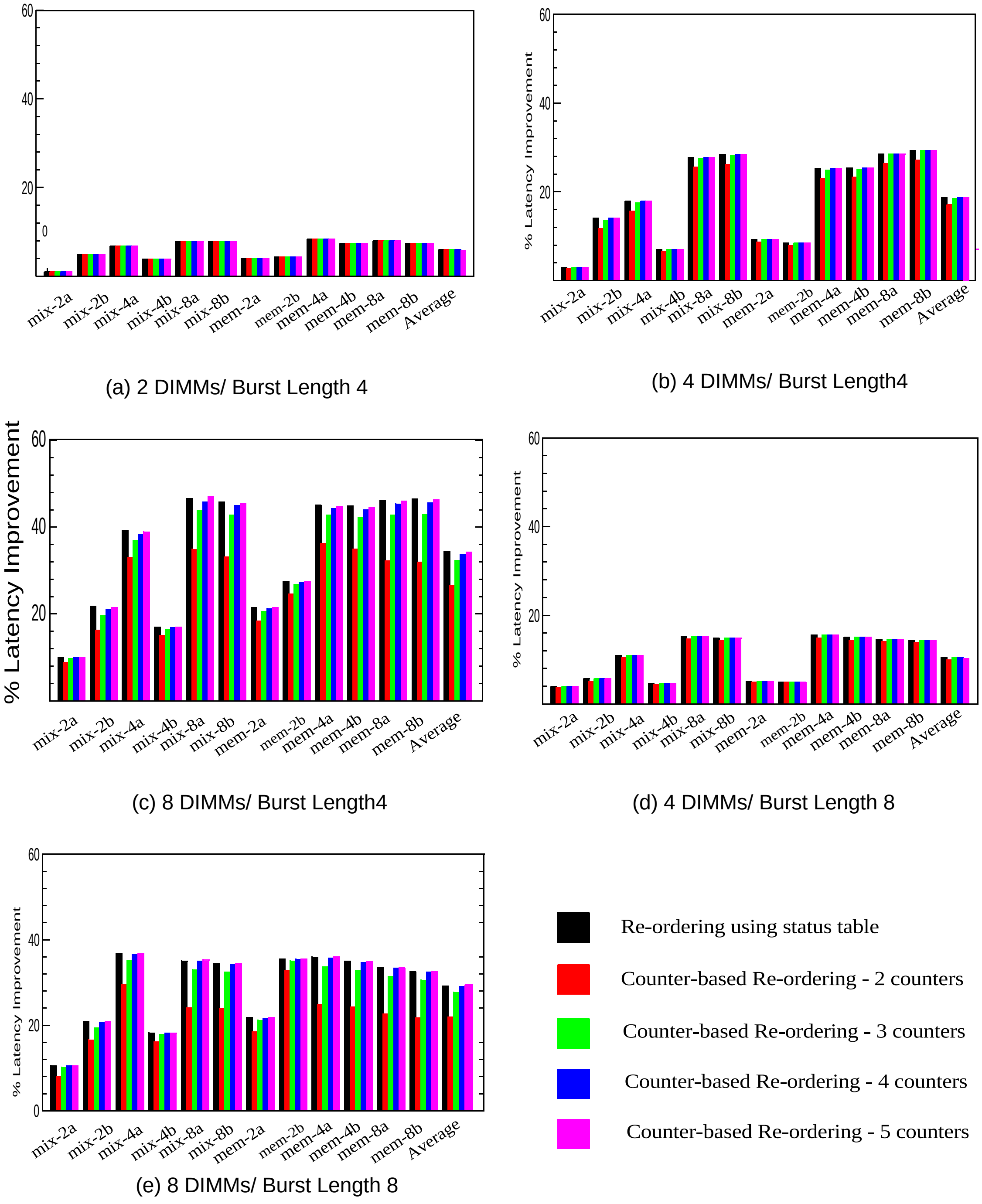
maximum number of counters which are required to track the link usage are three, we find that the opportunities to do so are rare. This is because the scheduling policy is greedy and does not attempt to reorder transactions such that the return link is 100% utilized.

5.2.4 Re-ordering Data Returns : Application Performance Impact

As expected permitting reordered data return of read data increases with the number of DIMMs in the channel. The average improvement in read latency is around 20-35% for systems with 8 DIMM deep channels, 10-20% for 4 DIMM deep channels and 0-5% for 2 DIMM deep channels. Improvements in latency are a function of the burst length and channel depth, with the improvements decreasing as we go to longer burst lengths and shorter channel lengths. The improvements decrease in a system with more bandwidth, but are independent of the row buffer management policy. FBD-DDR3 systems benefit more than FBD-DDR2 systems because reordering is able to mask their higher transmission costs.

Figure 5.26 and 5.27 show the improvements in read latency by permitting transactions to returned in a different order from which they were issued in for FBD-DDR3 systems using a closed page policy and with one and two channels of memory respectively. Figure 5.28 shows the improvements for single channel open page FBD-DDR3 systems, while figure 5.29plots the same for closed page FBD-DDR2 systems.

By keeping track of link usage on a per-cycle basis, the status table has the most information of all the schemes on link usage. Hence, status table based reordering represent the upper bounds in improvement for a greedy re-ordering scheduling policy. On the other hand, the counter-based mechanism represents a less complex and simpler solution to track link usage and is sufficient to extract all the performance improvements available by per-



(a) 2 DIMMs/ Burst Length 4

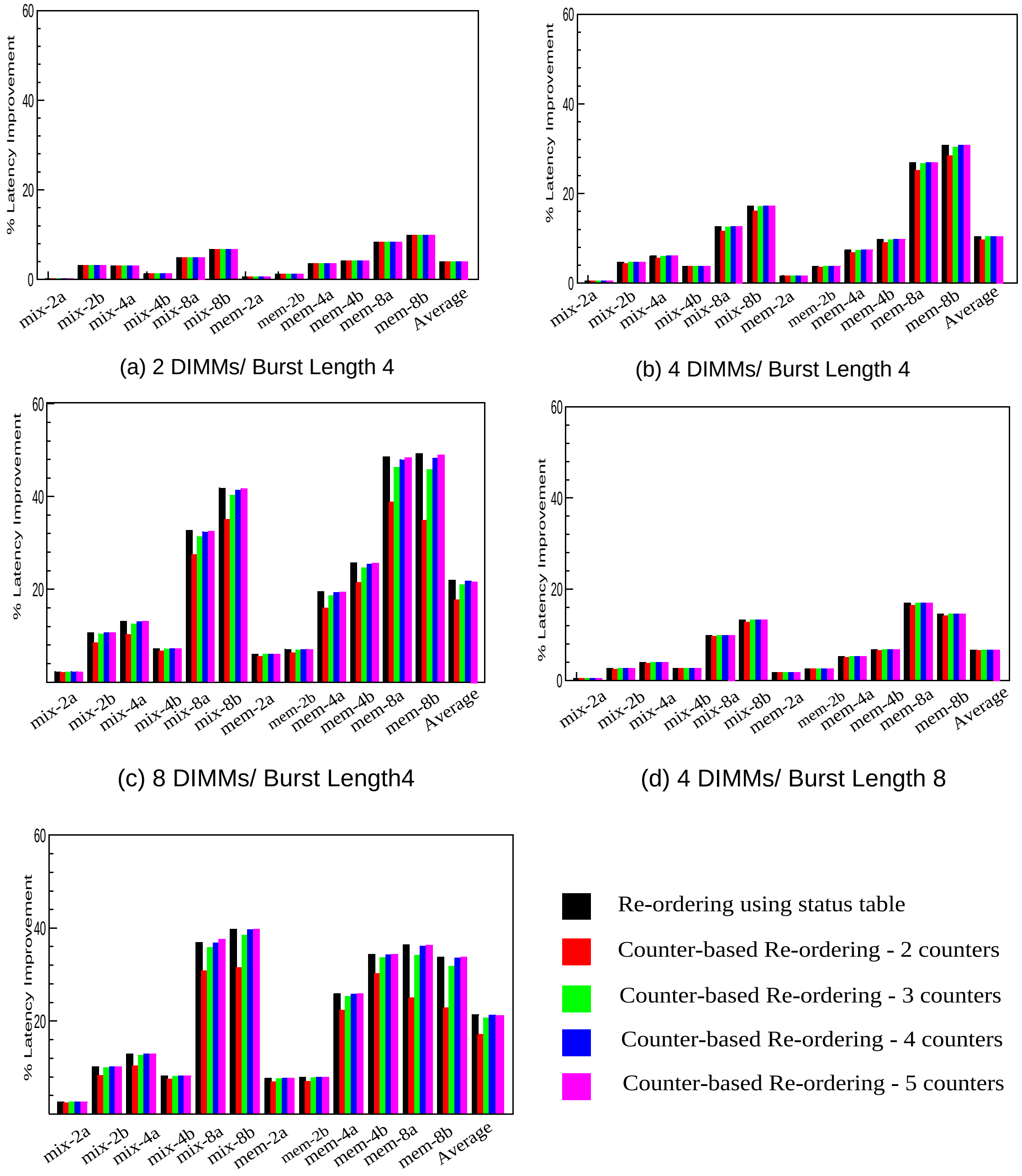
(b) 4 DIMMs/ Burst Length4

(c) 8 DIMMs/ Burst Length4

(d) 4 DIMMs/ Burst Length 8

(e) 8 DIMMs/ Burst Length 8

Figure 5.26. Improvements in Latency by permitting out-of-order return in FBD-DDR3 system using closed page single channel. The figure shows the improvements in permitting read data to return out of order for different applications at different channel depths. On the x-axis the data for each application is provided. Each application's data comprises of 4 bars each for a different re-ordering scheme implementations.



(a) 2 DIMMs/ Burst Length 4

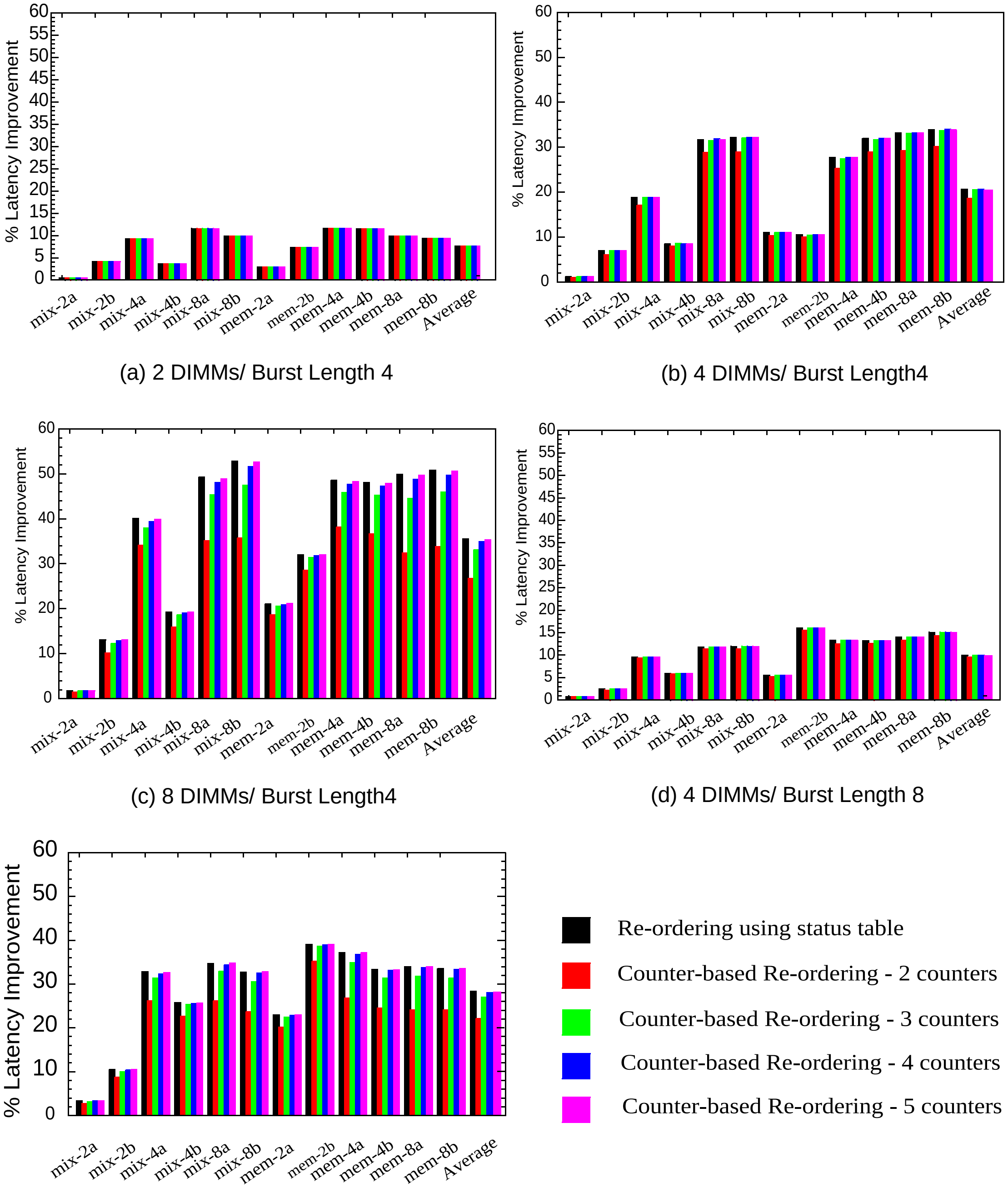
(b) 4 DIMMs/ Burst Length 4

(c) 8 DIMMs/ Burst Length4

(d) 4 DIMMs/ Burst Length 8

(e) 8 DIMMs/ Burst Length 8

Figure 5.27. Improvements in Latency by permitting out-of-order return in FBD-DDR3 system using closed page in a two channel system. The figure shows the improvements in permitting read data to return out of order for different applications at different channel depths. On the x-axis the data for each application is provided. Each application's data comprises of 4 bars each for a different re-ordering scheme implementations.



(a) 2 DIMMs/ Burst Length 4

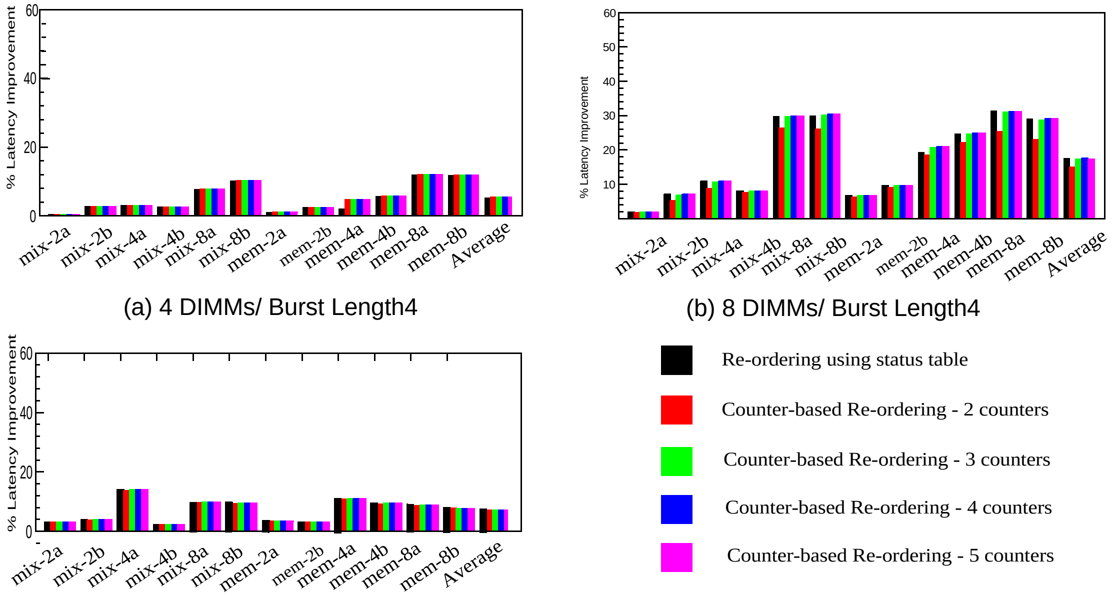
(b) 4 DIMMs/ Burst Length4

(c) 8 DIMMs/ Burst Length4

(d) 4 DIMMs/ Burst Length 8

(e) 8 DIMMs/ Burst Length 8

Figure 5.28. Improvements in Latency by permitting out-of-order return in FBD-DDR3 system using open page in a single channel system. The figure shows the improvements in permitting read data to return out of order for different applications at different channel depths. On the x-axis the data for each application is provided. Each application's data comprises of 4 bars each for a different re-ordering scheme implementations. Note that the y-axis on each of the graphs is different.



(a) 4 DIMMs/ Burst Length4

(b) 8 DIMMs/ Burst Length4

(c) 8 DIMMs/ Burst Length 8

Figure 5.29. Improvements in Latency by permitting out-of-order return in FBD-DDR2 system using closed page in a single channel system. The figure shows the improvements in permitting read data to return out of order for different applications at different channel depths. On the x-axis the data for each application is provided. Each application's data comprises of 4 bars each for a different re-ordering scheme implementations. Note that the y-axis on each of the graphs is different.

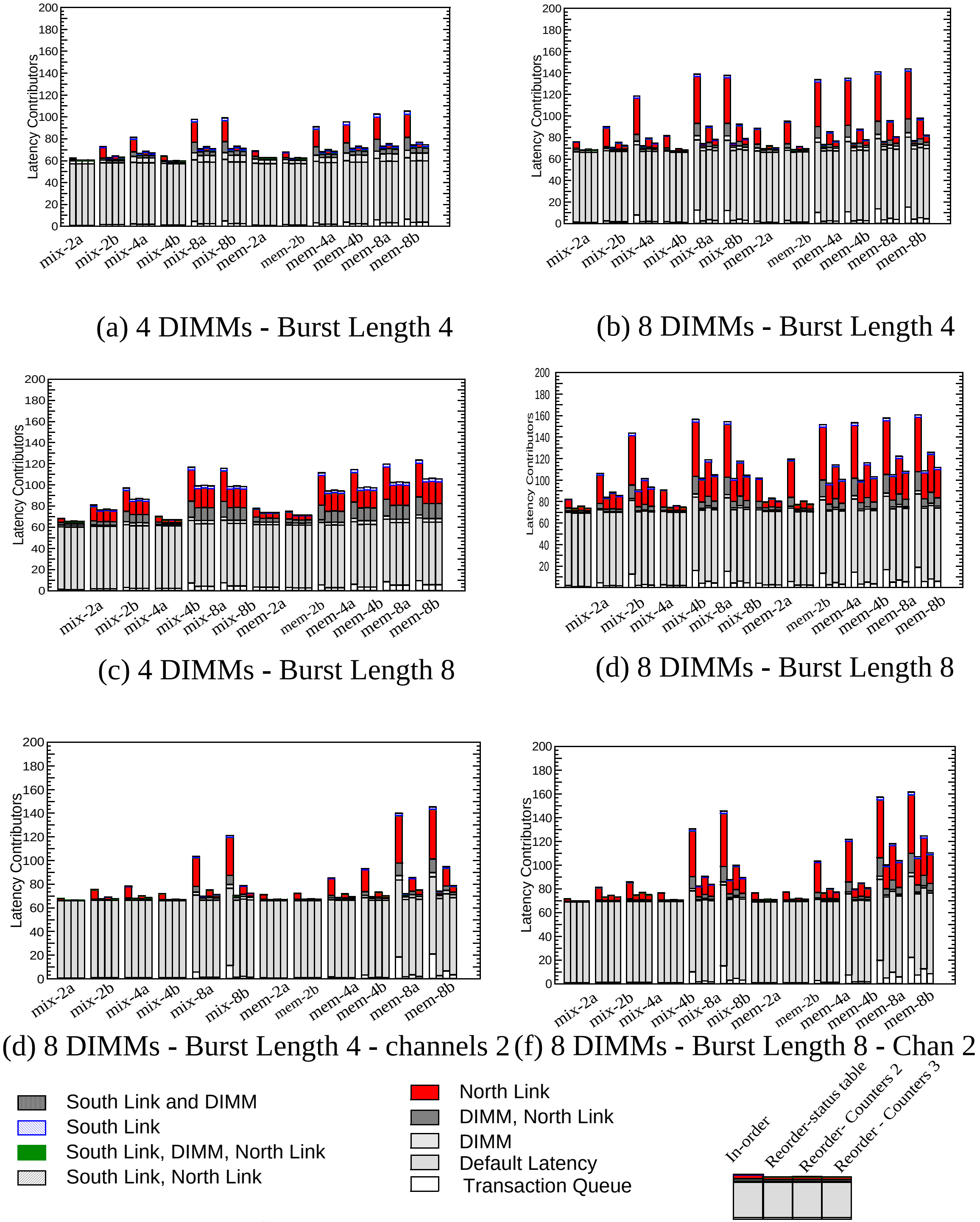
mitting re-ordering. By tracking the last two north link usage times latency reductions which are within 50-100% of that achieved by using a status table. In 2 and 4 deep channel systems (Fig 5.26 (a - c)), tracking the last two link use times is sufficient to capture all the possible re-ordering possible. However, in an 8 DIMM deep channel (Fig 5.26 (d, e)) tracking the last three link uses is sufficient to achieve all the latency reductions possible and using two counters results in 50-70% of the maximum possible gain.

Using reordering is beneficial in systems where the idle gaps between two read data bursts is large enough to accommodate another burst. We find that this is true for nearly all channel lengths and burst lengths in a FBD-DDR3 system except a 2 DIMM deep channel in a system operating with burst length 8. However, for an FBD-DDR2 system, Fig 5.29, re-ordering is effective only in a system with at least 4 DIMMs per channel. This is because the transmission costs, which are determined by the speed of transmission of the electrical signal on the wires and is independent of channel speed, takes fewer clock cycles in a FBD-DDR2 system. Consequently FBD-DDR2 systems start seeing benefits at longer channel lengths.

Benefits from re-ordering are higher for applications which have higher bandwidth utilization, such as the workloads with 4 and 8 active threads. For shorter channel lengths we find that the workloads with two threads do not benefit at all from re-ordering (Fig 5.29(a)). This is because requests are spaced sufficiently far apart that no requests to earlier DIMMs get delayed by outstanding requests to DIMMs which are further from the memory controller.

Open page systems benefit from permitting out of order data returns as much as closed page systems do (Fig 5.26 and Fig 5.28). This is not surprising because the benefits come from taking re-ordering based on north link availability and not DRAM bank conflicts. We found that in systems with more bandwidth (Fig 5.27) benefit less from re-ordering than systems with lesser available channel bandwidth (Fig 5.26). This is because as seen earlier, more channel level parallelism reduces the north link bottleneck.

Figure 5.30 shows the variation in the contributors to average read latency due to the re-ordering of data returns for a FBD-DDR3 closed page system. The graph plots the average latency, split into its various contributors, on the y-axis for each application. On the x-axis the bars are grouped based on the workload being studied. Within each group, each bar is representative of a different policy regarding re-ordering data returns on the northbound channel. The first bar represents a configuration where all data returns in order and the next three are for configurations where re-ordering of data returns are permitted each with a dif-



(a) 4 DIMMs - Burst Length 4

(b) 8 DIMMs - Burst Length 4

(c) 4 DIMMs - Burst Length 8

(d) 8 DIMMs - Burst Length 8

(d) 8 DIMMs - Burst Length 4 - channels 2

(f) 8 DIMMs - Burst Length 8 - Chan 2

Figure 5.30. Impact of Re-ordering Data Returns on Latency Contributors. he graphs show the contributors to average read latency for a transaction in the system for different applications. Each graph shows data for a system configuration with a particular channel depth, number of channels and burst length. On the x-axis of each group, the data for each workload combination is shown. Within each workload combination, each bar represents a different re-ordering scheme.

ferent method for tracking the north link usage information. The first in the latter set tracks north link usage using a status table and the remaining use counters to do so.

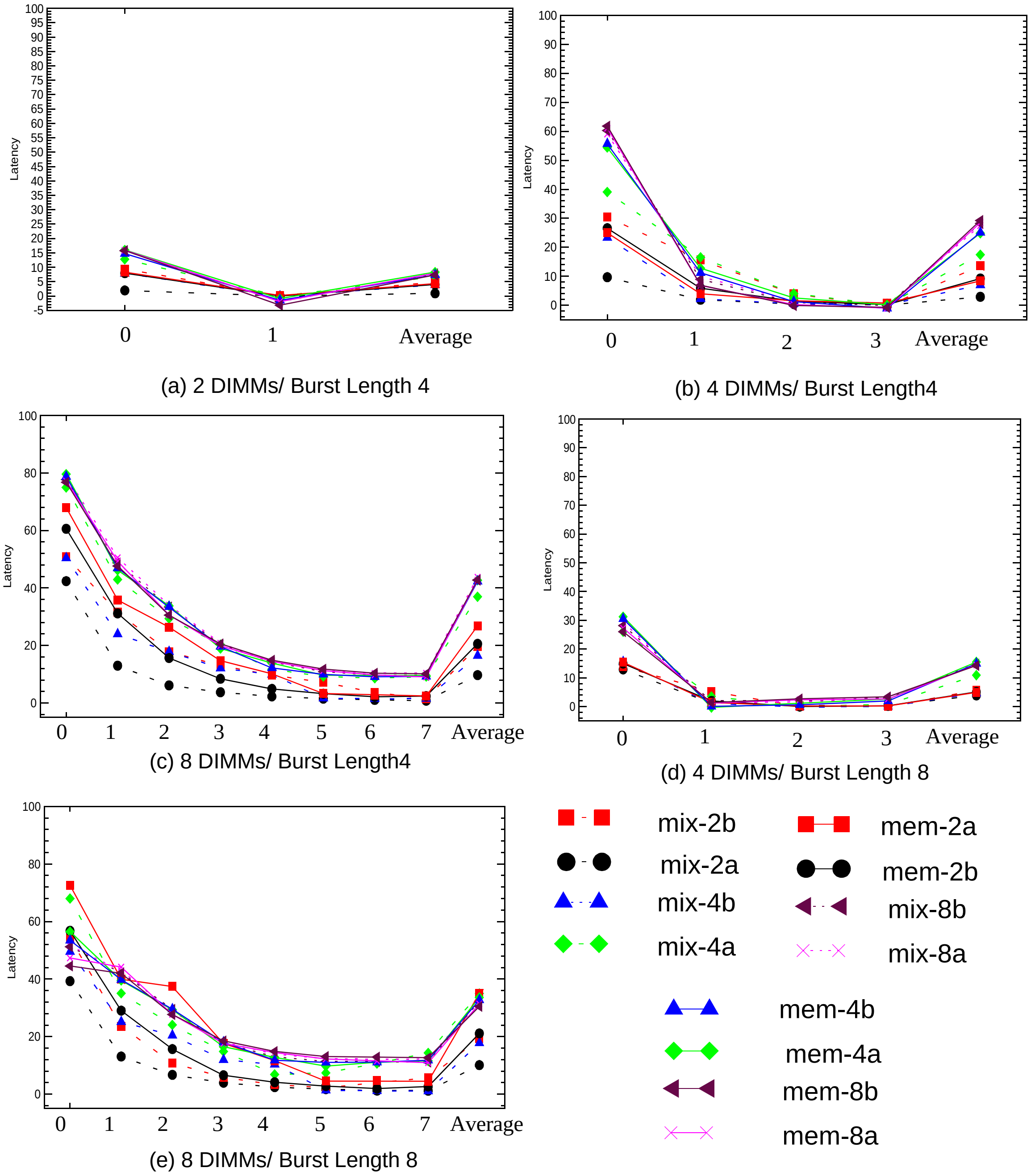
As expected, the bulk of the reductions in latency are achieved due to a reduction in queueing delay due to the north link being unavailable. In the case of burst length 4 systems, the north link queueing delay is eliminated for nearly all workloads by re-ordering data returns (Figure 5.30 a, b). However, the reductions in north link queueing delay are a function of the northbound link usage tracking mechanism. In a 8 DIMM deep channel, tracking the northbound channels using a status table is more effective than counter based tracking (Figure 5.30b). Although in the latter case, increasing the number of counters results in an increased reduction in northbound link queueing delay it is not completely eliminated for even 3 counters.

In the case of a burst length 8 system (Figure 5.30 (c, d), we found that using re-ordering reduces northbound link associated queueing delay but does not eliminate it. The consequence of the longer burst length is that the link utilization is higher and transactions are delayed waiting for a data burst to complete. Adding an extra channel (Figure 5.30 e) to the system increases the bandwidth available to the application and results in a further reduction in the queueing delay due to the northbound link being unavailable.

Note than in addition to the queuing delay associated with the unavailability of the northbound channel, additional reductions in read latency are achieved due to a reduction in the queueing delay associated with the transaction queue being unavailable. This is especially true for the mem-8 workloads (Figure 5.30 b, e). Re-ordering data returns on the northbound channel results in transactions moving out of the transaction queue faster and consequently freeing up this queue for new transactions which are waiting in the BIU for a free slot in the read transaction queue.

The reductions in overall read latency were achieved mainly by reducing the read latency for transactions which were addressed to earlier DIMMs in the chain. Fig 5.31 plots the improvement in average read latency experienced by a transaction being sent to a particular DIMM. The reductions in read latency vary from 10-80% for read transactions addressed to the first DIMM in the chain to 0-10% for transactions addressed to the last DIMM in the chain. For all channel lengths, the reductions in read latency were highest for transactions to the DIMM closest to the memory controller. The performance improvements drop sharply with the distance of the DIMM from the memory controller till it reaches zero for the last DIMM in the channel (Fig 5.31 a, b, d).However, for an 8 DIMM deep channel, we find that permitting re-ordering of data returns results in a reduction in latency for even transactions to the last few DIMMs in the channel (Fig 5.31 c, f). These reductions were achieved due to the reduction in queueing delay to the transaction queue being unavailable.

An examination of the latency contributors to the average read latency experienced by a transaction to a particular DIMM (Fig 5.32 and 5.33) shows that the reductions in queueing delay for transactions to the first few DIMMs in the channel were significant. This reduction in latency is largest for the first DIMM in the chain. In the case of workloads with 4 and 8 threads, we see that some of this queueing delay still remains. Fig 5.32 (c, d, e and f) and 5.33 (c, d, e and f) both show the reduction in queueing delay due to the transaction queue being unavailable resulting in the reduction in read latency for transactions to the last DIMM in an 8 DIMM system.



(a) 2 DIMMs/ Burst Length 4

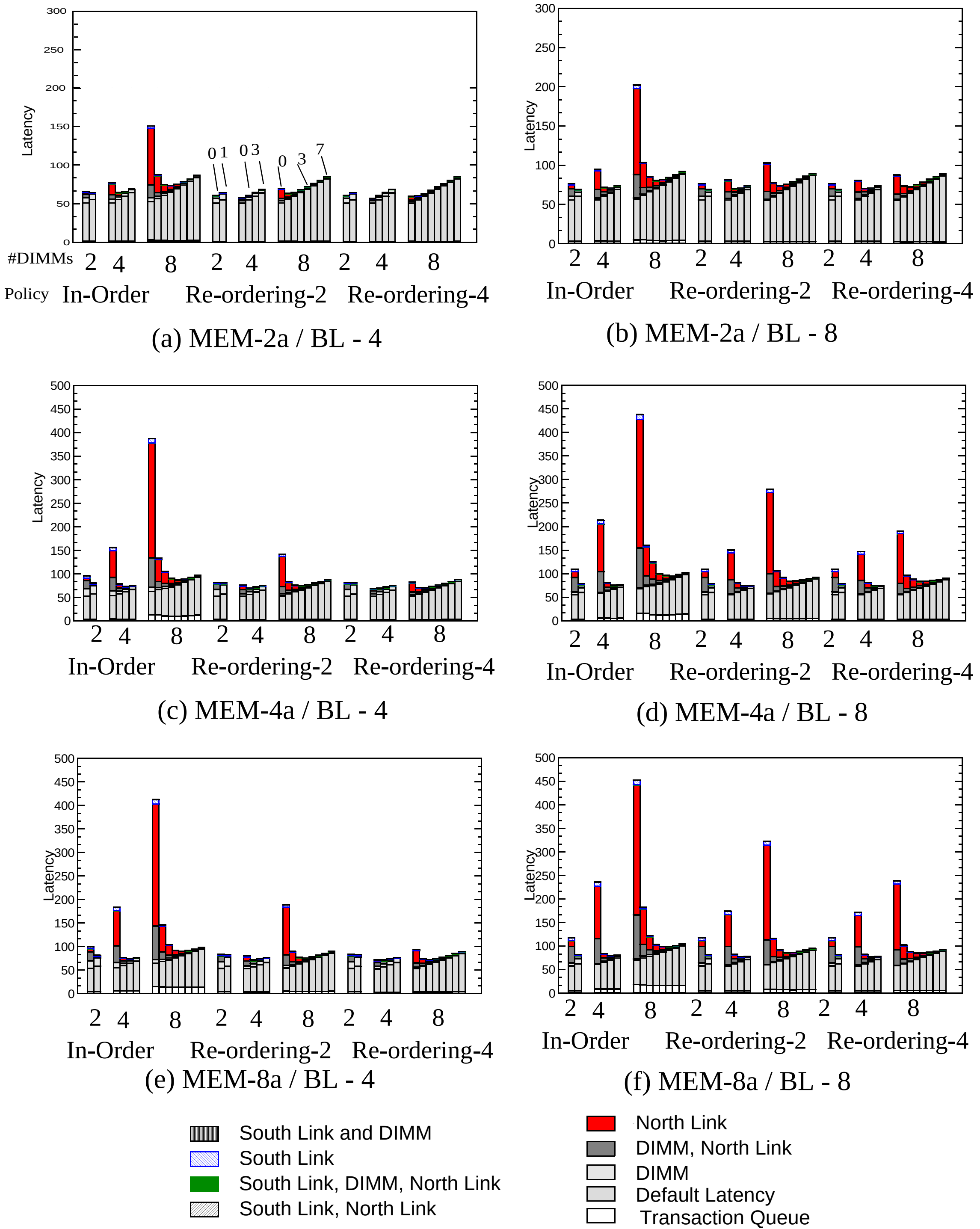
(b) 4 DIMMs/ Burst Length4

(c) 8 DIMMs/ Burst Length4

(d) 4 DIMMs/ Burst Length 8

(e) 8 DIMMs/ Burst Length 8

Figure 5.31. Per-DIMM Latency Improvements by a Re-ordering Data Returns. he graphs show the contributors to average read latency for a transaction in the system for different applications. Each graph shows data for a system configuration with a particular channel depth, number of channels and burst length. On the x-axis of each group, the data for each workload combination is shown. Within each workload combination, each bar represents a different re-ordering scheme.



(a) MEM-2a / BL - 4

(b) MEM-2a / BL - 8

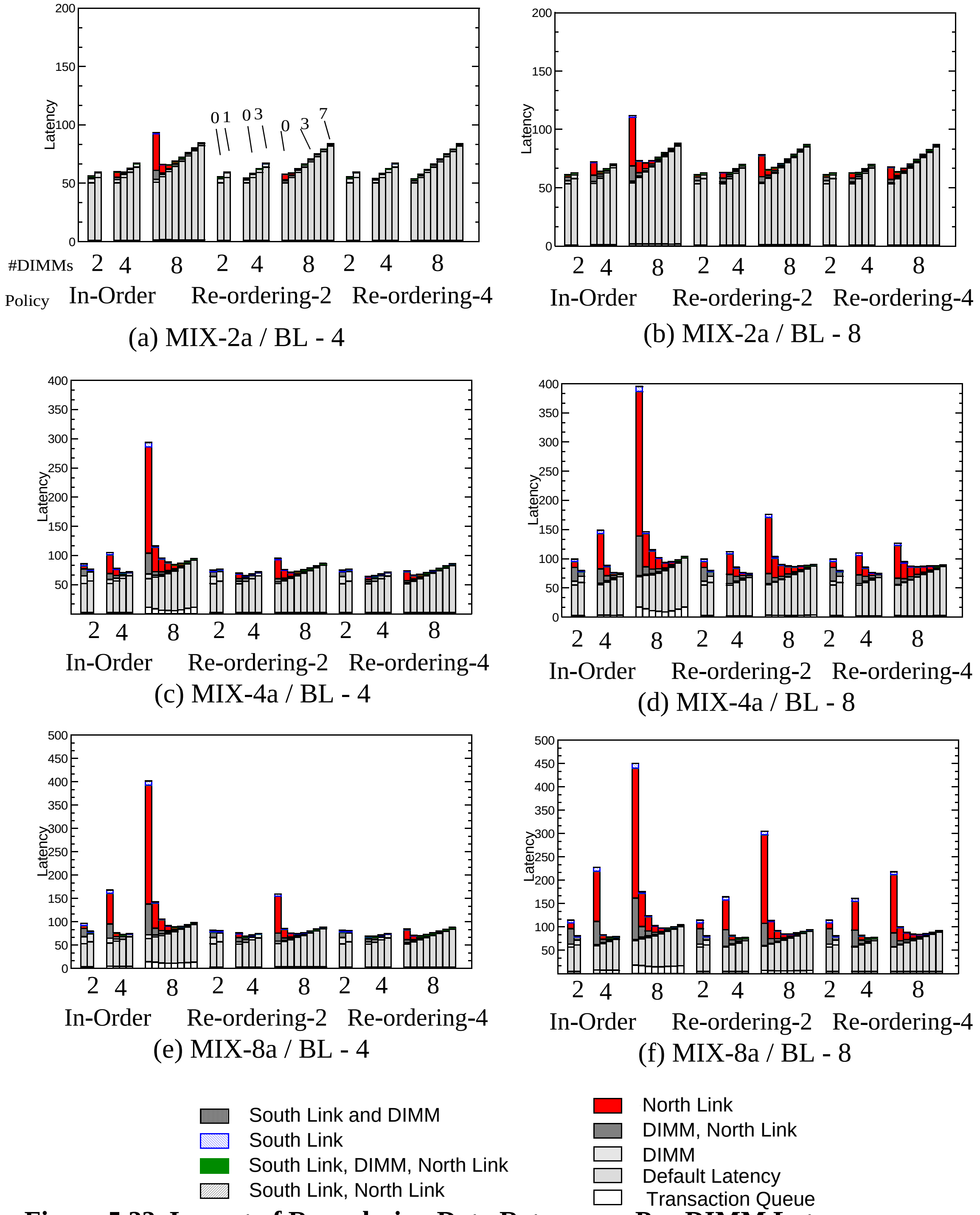
(c) MEM-4a / BL - 4

(d) MEM-4a / BL - 8

(e) MEM-8a / BL - 4

(f) MEM-8a / BL - 8

Figure 5.32. Impact of Re-ordering Data Returns on Per-DIMM Latency Contributors. he graphs show the contributors to average read latency for a transaction to a given DIMM for different applications. The x-axis plots the per-DIMM latency distribution for different re-ordering schemes for different channel depths. The ranks are ordered such that the DIMM closest to the memory controller has the lowest number. The configurations are grouped on the x-axis such that the leftmost group his a channel with 2 ranks while the rightmost has 8 ranks.



(a) MIX-2a / BL - 4

(b) MIX-2a / BL - 8

(c) MIX-4a / BL - 4

(d) MIX-4a / BL - 8

(e) MIX-8a / BL - 4

(f) MIX-8a / BL - 8

Figure 5.33. Impact of Re-ordering Data Returns on Per-DIMM Latency Contributors. he graphs show the contributors to average read latency for a transaction to a given DIMM for different applications. The x-axis plots the per-DIMM latency distribution for different re-ordering schemes for different channel depths. The ranks are ordered such that the DIMM closest to the memory controller has the lowest number. The configurations are grouped on the x-axis such that the leftmost group his a channel with 2 ranks while the rightmost has 8 ranks.

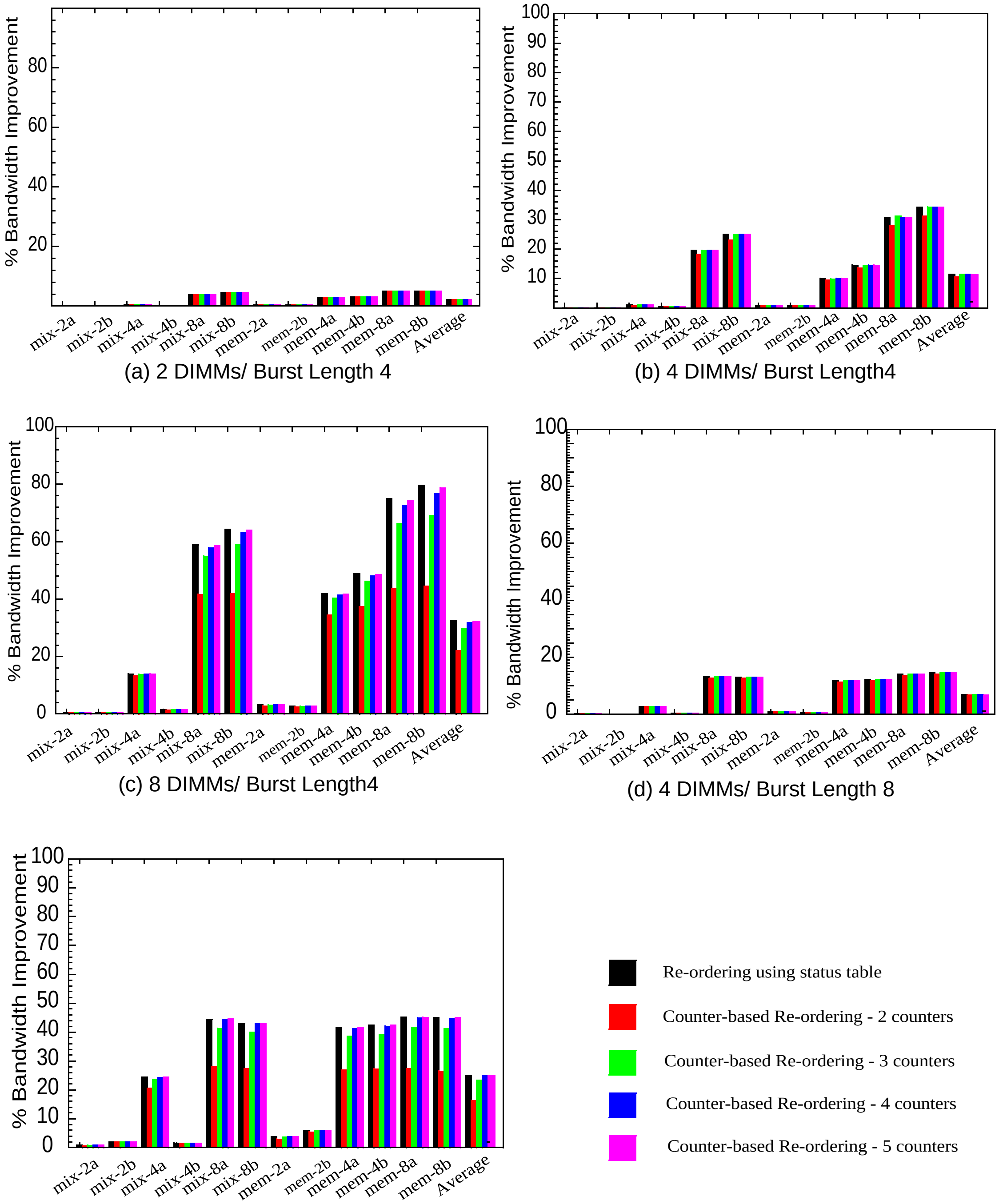
5.2.4.1 Bandwidth Improvement

As in the case of latency, bandwidth improvements were a function of the burst length, the number of DIMMs in the channels and the channel utilization. We saw that the bandwidth improvements were highest for an 8 DIMM deep channel which uses a burst length of 4. Although average bandwidth improvements were in the same range as latency improvements, average of 10-40% for 2-8 DIMM deep channels, we see that some workloads experienced as much as and 80% improvement in bandwidth.

Fig 5.34 and 5.35 plot the improvements in bandwidth for various channel lengths for a closed page FBD-DDR3 with one and two channels of memory respectively. Fig 5.36 plot the improvements for an FBD-DDR3 open page system while fig 5.37 plots the improvements for a closed page FBD-DDR2 system.

As in the case for latency improvements, we found that tracking link usage by using a status table yielded the highest improvements possible and represented the upper bound for performance improvements possible. Unlike the case of latency improvements, we found that using only 2 counters to track the last two uses of the northbound link resulted in typically 50% of the maximum possible improvements in bandwidth for the 4 and 8 way applications (Fig 5.34 (c, e) and Fig 5.36 (c, e)) in a channel with 8 DIMMs. Tracking the last three recovered the bulk of the remaining savings and we find that the improvements in bandwidth saturated with further increases in the number of north link last uses tracked.

In the case of the workloads with 8 threads, which have a higher request rate, we see that the bandwidth utilization improves by nearly 80-90% (Fig 5.34 c and Fig 5.36 c). This



(a) 2 DIMMs/ Burst Length 4

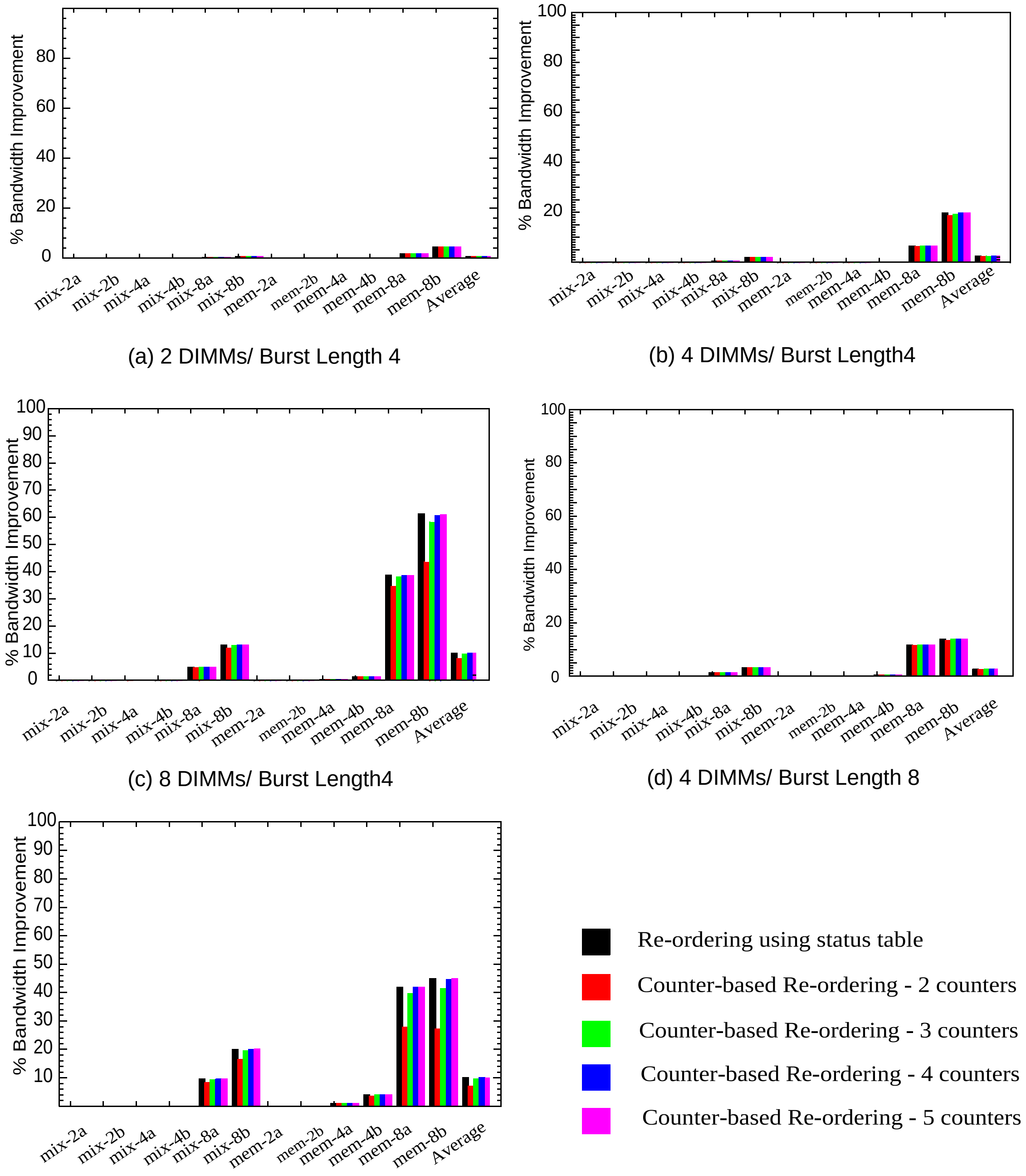
(b) 4 DIMMs/ Burst Length4

(c) 8 DIMMs/ Burst Length4

(d) 4 DIMMs/ Burst Length 8

(e) 8 DIMMs/ Burst Length 8

Figure 5.34. Improvements in Bandwidth by permitting out-of-order return in FBD-DDR3 system using closed page in a single channel system. The figure shows the improvements in permitting read data to return out of order for different applications at different channel depths. On the x-axis the data for each application is provided. Each application's data comprises of 4 bars each for a different re-ordering scheme implementations. Note that the y-axis on each of the graphs is different.



(a) 2 DIMMs/ Burst Length 4

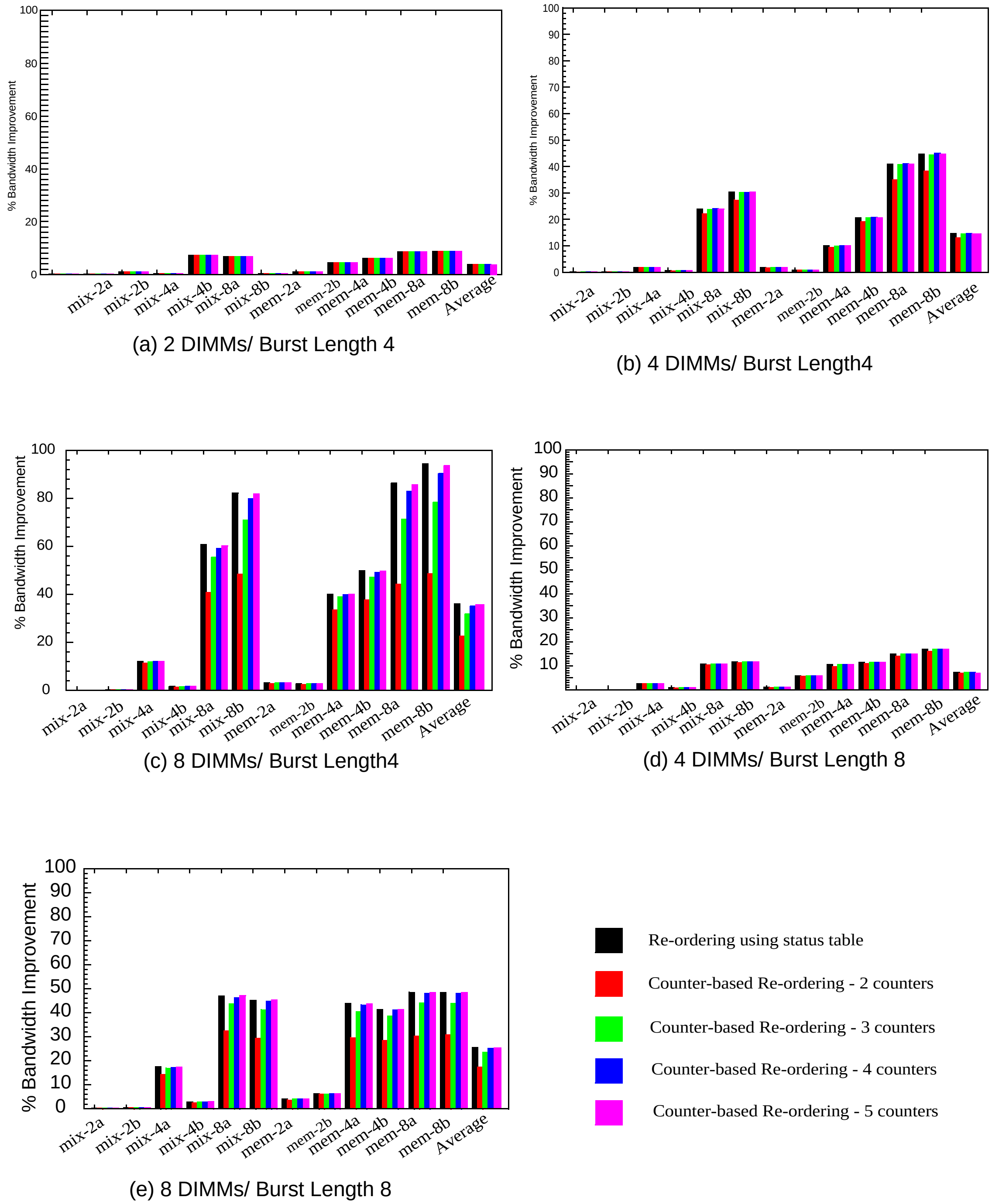
(b) 4 DIMMs/ Burst Length4

(c) 8 DIMMs/ Burst Length4

(d) 4 DIMMs/ Burst Length 8

(e) 8 DIMMs/ Burst Length 8

Figure 5.35. Improvements in Bandwidth by permitting out-of-order return in FBD-DDR3 system using closed page in a two channel system. The figure shows the improvements in permitting read data to return out of order for different applications at different channel depths. On the x-axis the data for each application is provided. Each application's data comprises of 4 bars each for a different re-ordering scheme implementations. Note that the y-axis on each of the graphs is different.



(a) 2 DIMMs/ Burst Length 4

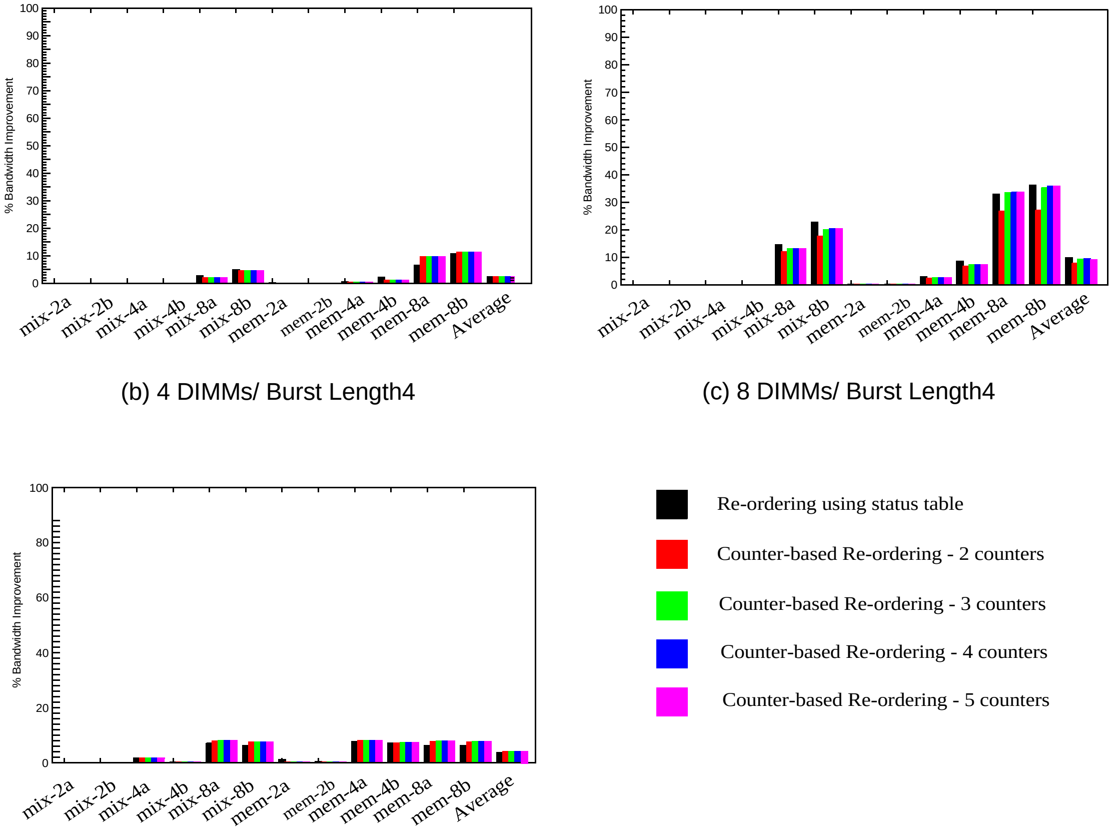
(b) 4 DIMMs/ Burst Length4

(c) 8 DIMMs/ Burst Length4

(d) 4 DIMMs/ Burst Length 8

(e) 8 DIMMs/ Burst Length 8

Figure 5.36. Improvements in Bandwidth by permitting out-of-order return in FBD-DDR3 system using open page in a single channel system. The graphs plot the percentage improvements for the observed bandwidth on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with each bar for different re-ordering scheme implementations. Note that the y-axis on each of the graphs is different.



(b) 4 DIMMs/ Burst Length4

(c) 8 DIMMs/ Burst Length4

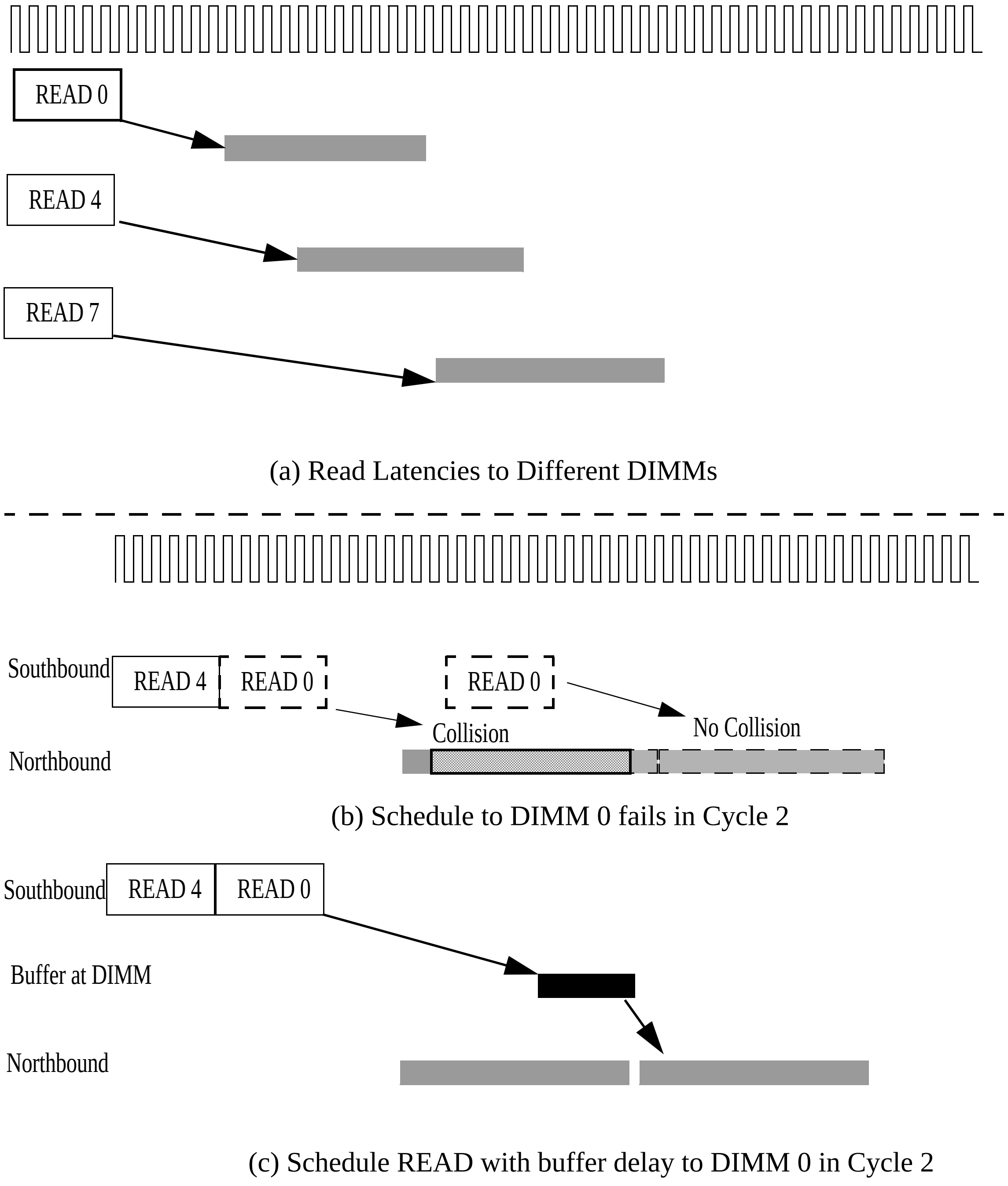
(e) 8 DIMMs/ Burst Length 8

Figure 5.37. Improvements in bandwidth by permitting out-of-order return in FBD-DDR2 system using closed page in a single channel system. The graphs plot the percentage improvements for the observed bandwidth on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with each bar for different re-ordering scheme implementations. Note that the y-axis on each of the graphs is different.

huge improvement in bandwidth is because of the increased usage of the bus possible by using the idle gaps present in the channel.

5.3. Buffering on the AMB

In this section we look at how buffering the return data frame at the AMB can be used to improve channel utilization and overall read latency. Buffering allows the memory controller to move data out of the DRAMs without having to wait for the north link to become available. Buffering improves DRAM level parallelism by releasing DRAM resources ear-



(a) Read Latencies to Different DIMMs

(b) Schedule to DIMM 0 fails in Cycle 2

(c) Schedule READ with buffer delay to DIMM 0 in Cycle 2

Figure 5.38. How buffering works. The top part of the figure shows the read latencices for read requests to different DIMMs in the same channel, numbered in increasing order as you move away from the memory controller. Part (b) shows that a read to DIMM 0 cannot be scheduled till 2 command cycles later without their being a collission on the bus. Figure c shows how the memory controller can schedule a read to DIMM 0 with a buffering delay. The AMB releases the data onto the DIMM as soon as the buffering duration is past.

lier especially at medium to higher bandwidth utilizations, when the north link is a significant bottleneck.

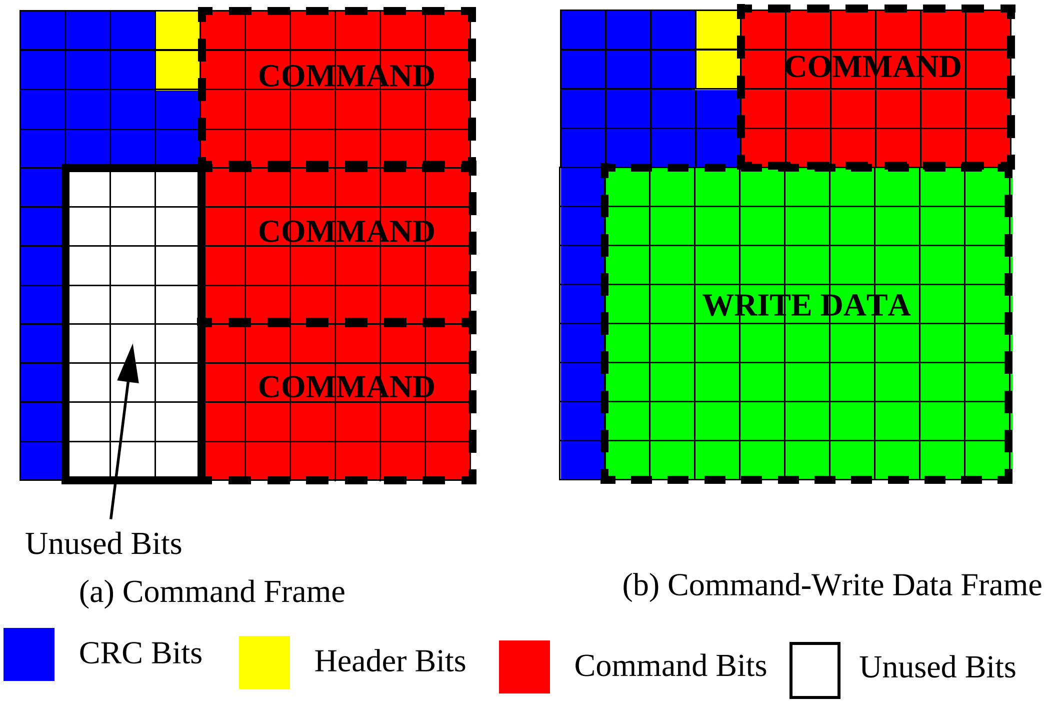
The FBDIMM AMB specification provides for limited AMB buffering. The AMB currently buffers the read data packet for a pre-specified duration determined at channel initialization. In the case of fixed latency mode of operation, this duration is specified such that it ensures that the round trip latency for a read transaction is identical for each DIMM in the system. In the variable latency mode of operation, the buffering duration is typically set to zero, thereby allowing the AMB to return its read data burst to the memory controller as soon as it is available and reduce read latency.

Figure 5.38 illustrates how buffering durations can be used to schedule requests to DIMMs even when the northbound link is not available. The read transaction suffers a slight buffering overhead but no further penalty waiting for other candidate transactions to complete, as in the case of the schedule in figure 5.15.

5.3.1 Buffering Techniques

5.3.1.1 Source Buffering

With this technique, the memory controller specifies the duration for which the read data should be buffered at the AMB of the DIMM supplying the data before sending it down to the memory controller. This is a fairly simple extension of current implementations of AMB buffering. With this technique, the pre-set buffering duration is modified to a variable one, with the duration specified with the CAS command. Note that in this case buffering is done only at the source AMB.



(a) Command Frame

(b) Command-Write Data Frame

Figure 5.39. FBDIMM Southbound Frame Bit Layout. The figure above shows the layout of the two main types of FBDIMM frames. Each southbound frame requires 12 transfers and is transmitted on a data bus which is 10 bits wide. Note that fail-over frames are similar except that they reduce the number of CRC bits

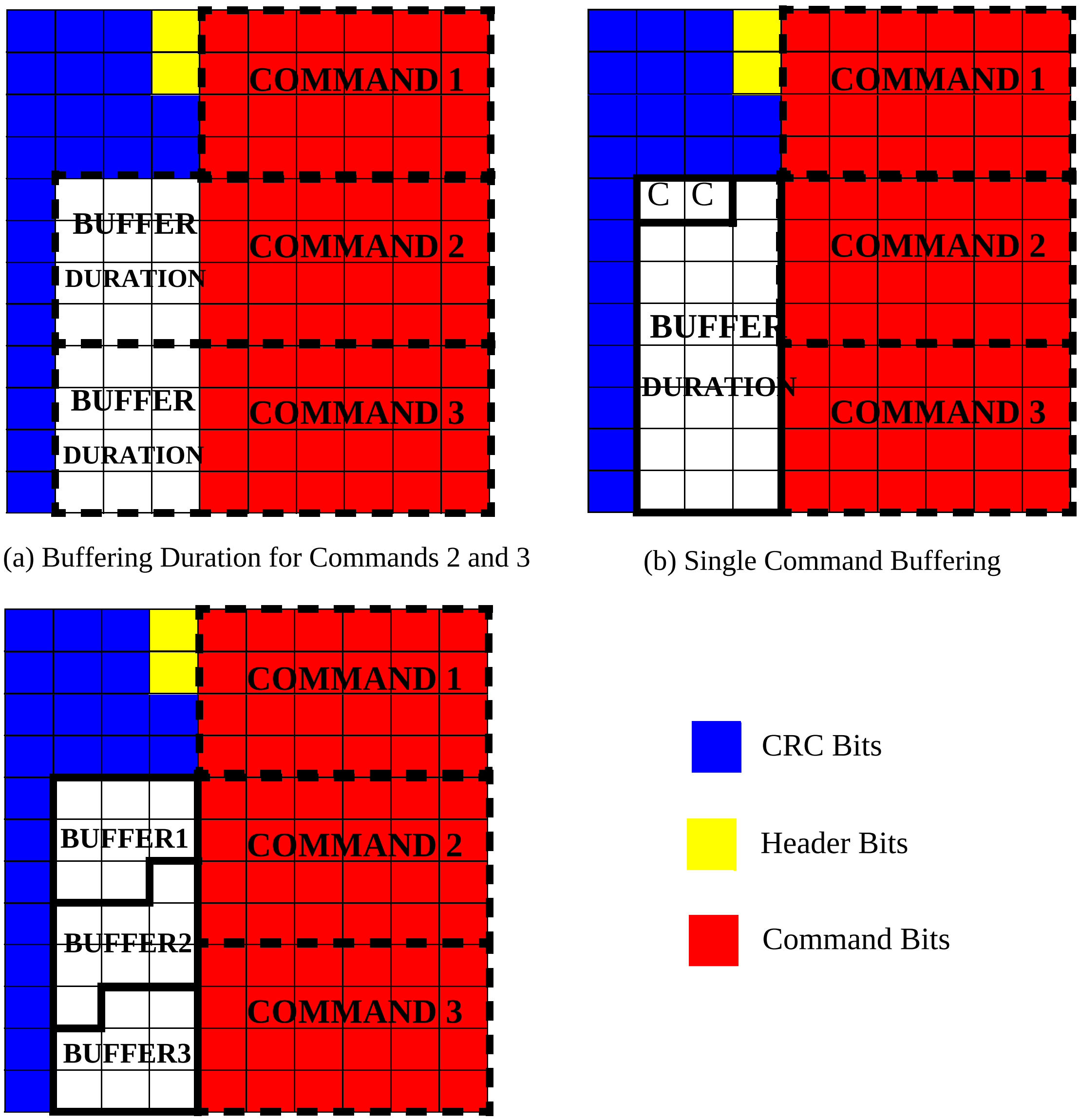
Southbound frames are of two types:- command-write-data frame and a command-only frame. In all cases the frame comprises of the payload (data or command information), header information and CRC. A FBDIMM command-write data frame has no free bits (Fig 5.39 (b)) while a command-only frame has 24 bits that are unused. In Fig 5.39 (a), the unused bits are clearly shown as unfilled squares.

The unused bits in the FBDIMM command-only frame can be used to specify the buffering duration. Since the unused bits are adjacent to the second and third command slot, the easiest approach would be to allocate the unused bits which are adjacent to a particular slot to specify the buffering duration for a command in that slot. This approach would restrict buffering capability only to CAS commands that are transmitted in the particular commands slots. The 12 bits can be used to specify a buffering duration of 0-4095 cycles. The buffering duration is specified in channel clock cycles for more fine-grained control. The bit layout for this scheme is shown in Fig 5.40 (a). This frame format fits easily into the frame definitions for a memory controller that uses posted CAS. In such a system, the RAS-CAS command pair are typically issued in the same frame, with the RAS in the first command slot and the CAS in the second command slot.

The alternate approach to permit would be use 2 command bits to specify the command slot associated with the buffering duration and use the remaining 22 bits to specify the buffer durations. Figure 5.40 (b) shows the possible layout of the command frame for such a case. The advantage of this approach would be that it would allow all CAS commands in the frame to be able to buffer. The disadvantage of this approach is that only one command in the frame would be capable of buffering.

Since there is no advantage in specifying buffering durations on the order of thousands of cycles, the third frame format can use the 24 bits to specify the buffering duration for all three commands in the system.Each command would have 8 bits to specify a buffering duration from 0-255 cycles. Note that in this case as well the decoding of the first command would take longer. Figure 5.40 (c) shows a possible layout of the FBDIMM command frame for this approach.

Implementation of buffering at source does not require significant modification to the AMB. Current implementations of the AMB support buffering the read data frame for a fixed duration of time, specified during channel initialization, before transmitting it. This scheme will first require modifying the decode logic to read the buffer duration. This infor-



(a) Buffering Duration for Commands 2 and 3

(b) Single Command Buffering

(c) Buffering Duration for All Commands

Figure 5.40. Command Frame Formats with Buffer Durations Specified. The above figures show the layouts for three different approaches to specifying buffer duration in a southbound command frame. The buffer duration is specified using the bits which are currently unused in a southbound command-only frame.

mation will then have to be used to program the buffering delay of the return data buffers.

Current AMB design uses a single buffer. We examine the impact of using multiple buffers

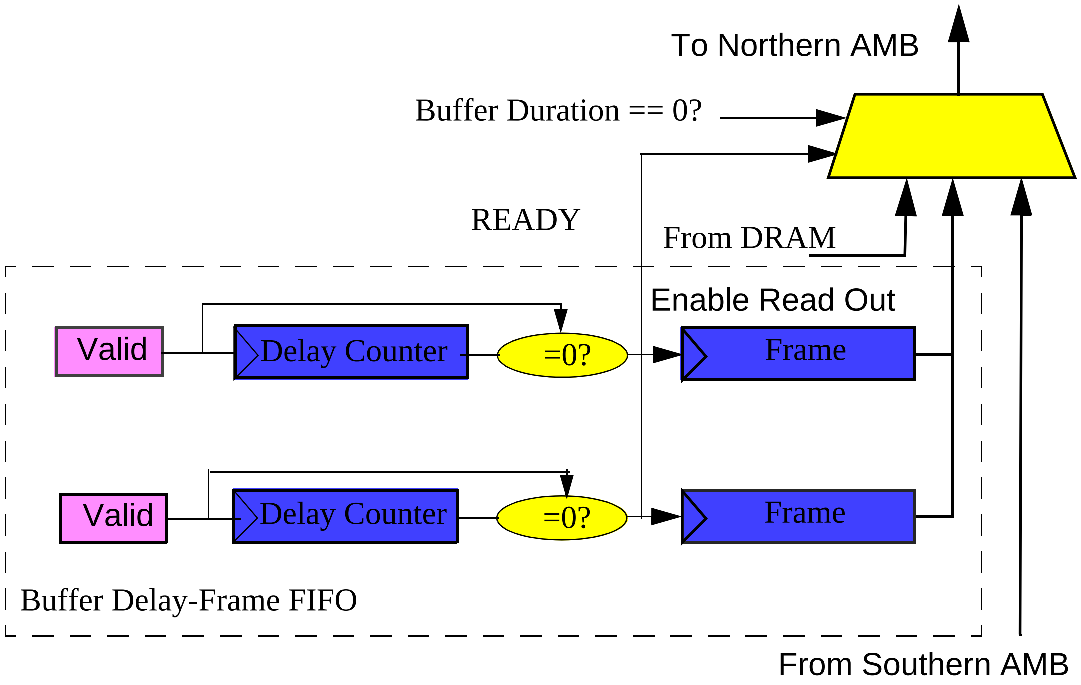


Figure 5.41. Hardware for Source Buffering. The figure above shows one possible implementation of the logic required to support buffering at source. The AMB return data path has to be modified to interpolate read data buffers with programmable delay counters.

which can be maintained in a FIFO queue structure. The hardware block diagram for the AMB return data path is shown in Fig 5.41.

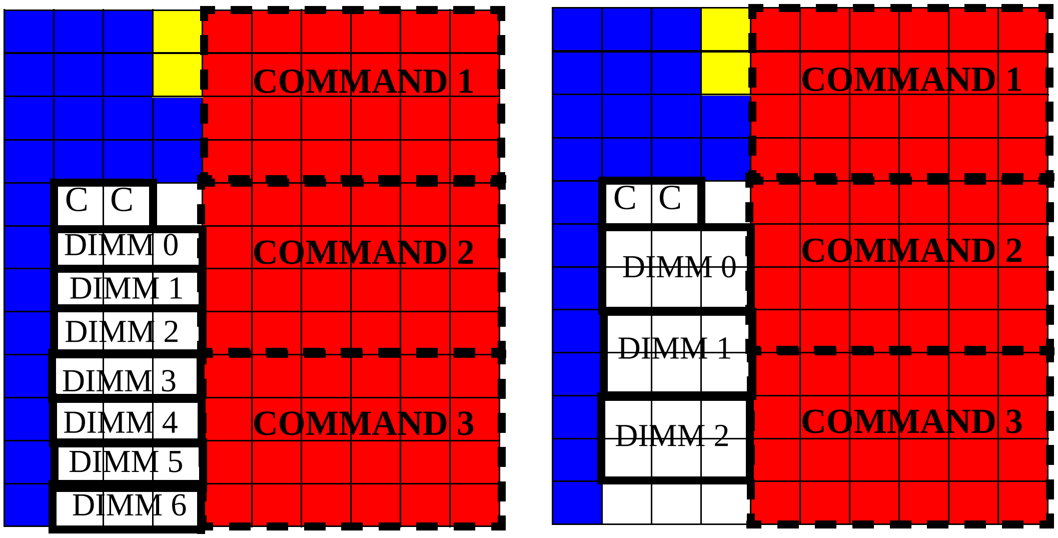
In terms of hardware implementation, each buffer entry comprises of a valid bit, a delay counter, a zero comparator and a buffer to hold the read data frame. Since a read command results in the creation of multiple return data frames, this buffer can be large enough to hold the entire read data burst. The buffering duration and read data frames are loaded to the tail of the FIFO queue. The read data frame will need to be loaded over multiple DRAM clock cycles. Loading the buffer duration causes the valid bit to be set and the delay counter to be enabled. The delay countdowns every clock cycle. When the delay value reaches zero, the frame is read out and transmitted on. To prevent data frames with zero buffering delay from being buffered a bypass path for such frames can be used.

5.3.1.2 Global Variable Buffering

Source Buffering is a fairly simple modification of the existing protocol. Its advantage is that it enables the memory controller to quickly schedule transactions to earlier DIMMs in the chain. This technique can however increase the queueing delay due to north link availability to DIMMs further south. Transactions to this DIMM may benefit from being buffered at multiple places. Hence, we study how the ability to specify buffering delays at multiple AMB locations will improve the overall read latency characteristics.

The second approach that we look at is to allow read data frames to be buffered at any AMB on its return path. As in the Source Buffering technique, each AMB has read buffers to store the read data frames that are to be sent back to the memory controller. Read data frames are stored in these buffers when the northbound link connected to these hubs is busy. The memory controller determines the duration for which a frame is delayed at every hub. Each buffer can hold all the frames to transmit a burst of data back to the memory controller.

The format of northbound command frames can be modified to specify the buffering duration at each link as shown in the figure. The number of bits available to specify the buffering duration per link varies depending on the total number of DIMMs in the channel. The deeper the channel the fewer the number of bits available, as can be seen in figure 5.42. Note that buffering durations are not specified for the last DIMM in the chain i.e. the DIMM furthermost away from the memory controller because read transactions to the last DIMM compete with only other transactions to the same DIMM for the northbound bus. The consequence of the serialization of requests to the same DIMM is that the last DIMM in the chain should experience no buffering delays. Note that two bits (marked C) are used



(a) Frame Layout for 8 DIMM channel (b) Frame Layout for 4 DIMM channe

Figure 5.42. Frames for Global Variable Buffering. The figure shows two different frame layouts for Global Variable Buffering. The number of bits allocated to specify the frame duration is a function of the number of DIMMs in the channel. Note that buffering duration is not specified for the last DIMM in the chain.

to specify which command in the frame the buffering durations are specified for. In this study, we ignore the restrictions on the number of bits required to specify the buffering duration for different DIMMs.

The AMB northbound data-path has to be modified to allow buffering of data frames both from its own DRAM as well as from its southern AMB. The AMB should be able to associate the buffering duration provided by the memory controller with the command with the appropriate read data frame. In a system that returns data frames in the same order that the commands were sent in this can be implemented by keeping a FIFO queue of buffering durations of all frames that are to pass through the AMB. When a read data frame arrives, the buffering duration is associated with the frame and it is delayed for the specified period

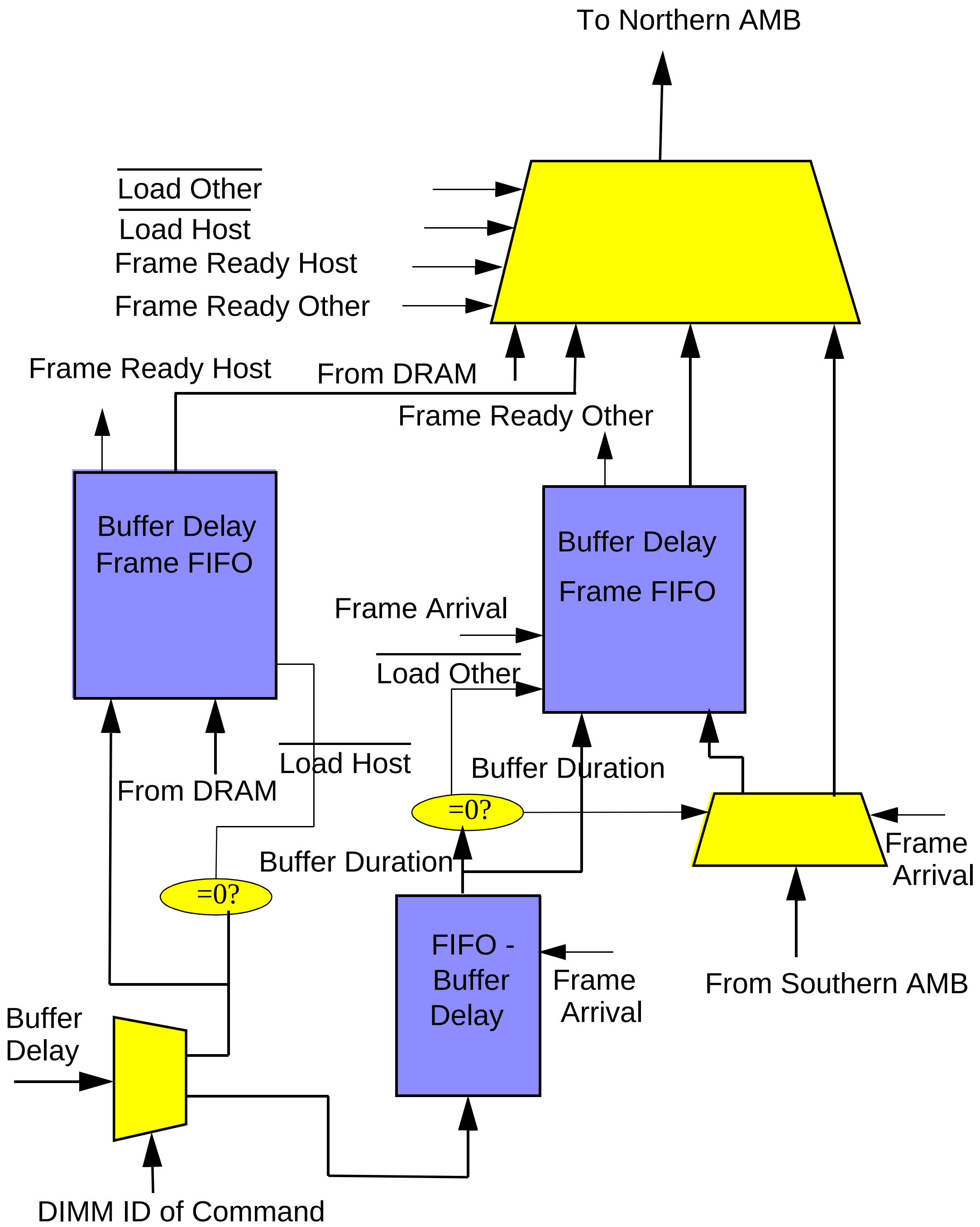


Figure 5.43. Hardware for Global Variable Buffering. The figure above shows one possible implementation of the logic required to support global variable buffering. The AMB return data path has to be modified to interpolate read data buffers with programmable delay counters for both frames from the AMB's own DRAM and from southern DIMMs. The queues holding the buffer delays and frames are similar to those sued in Source Buffering.

before being relayed on. To make the design simpler, the buffers and buffering duration queues for the two read data frame sources should be separate. The FIFO buffering queue used for the buffering at source can be used to buffer read data frames from the host DRAM. A similar buffering queue will be needed for the read data frames which are from the southern AMBs. Since the buffering delay and the read data frame arrive at the AMB at different times, the FIFO buffering queue for read data frames from different AMBs should have two different tail pointers, one for the buffer duration and the other for the read data frame.

5.3.1.3 Global Binary Buffering

The other option that we look at is Global Binary Buffering. This can be used with both systems which allow data to return in-order and out of order. In this scheme the buffer duration for the system is fixed at initialization by the memory controller. This buffer duration is the absolute time for which a transaction can be delayed. The memory controller uses a single bit to indicate whether a frame is to be delayed or not. Thus, the memory controller needs a maximum of 8 bits per command to indicate the buffering duration. The remaining bits can be used to indicate to the AMB the order in which the read data frame is to return with respect to the previously issued requests. The frame format for this particular approach would be as shown in figure 5.44. This scheme is a simplified implementation of global variable buffering.

5.3.2 Buffering: Impact on Application Performance

Buffering of return data frames improved the performance characteristics for all channel depths and burst lengths. The improvements were higher for longer channels with

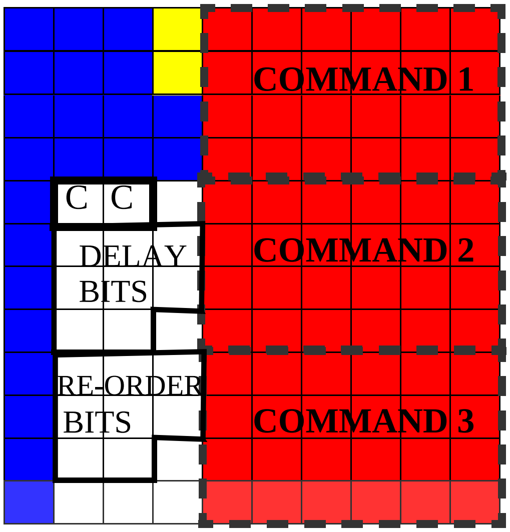


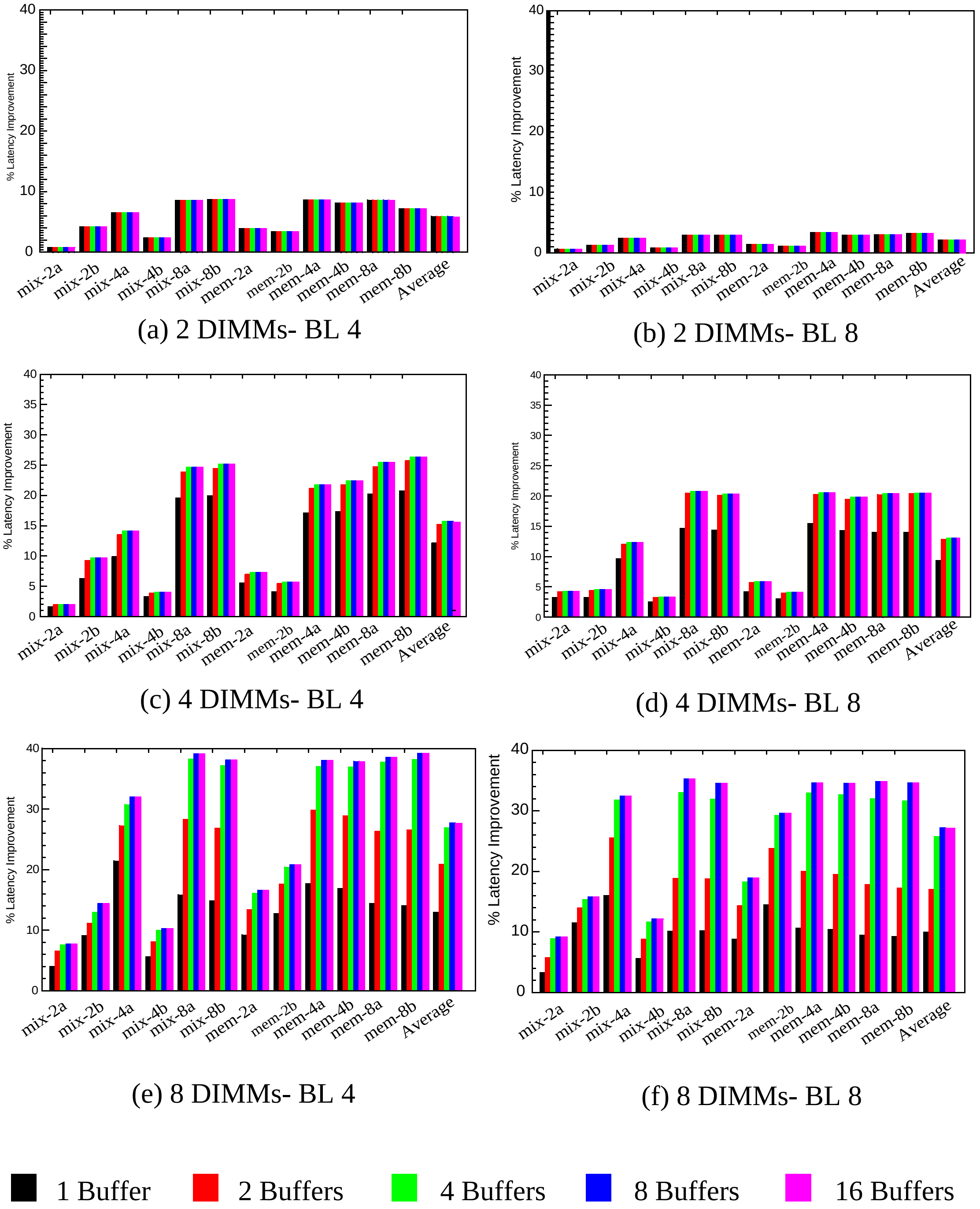
Figure 5.44. Frames for Global Binary Buffering. The figure shows two different frame layouts for Global Binary Buffering. The number of bits allocated to specify the frame duration is a function of the number of DIMMs in the channel. Note that buffering duration is not specified for the last DIMM in the chain.

average latency improvements across all workloads being 30% for an 8 DIMM deep channel, 15% for a 4 DIMM deep channel and around 5% for 2 DIMM deep channel. Performance improvements were seen for all channel depths and burst lengths but like reordering we found that burst length 8 systems benefitted less than burst length 4 systems from re-ordering. For channels with 4 or more DIMMs in the channel, we found that latency improved with the incorporation of additional buffers on the AMB for buffering return data up to a certain number of buffers, beyond which no further reductions in latency were observed. The buffer count beyond which no improvements were seen was a function with channel depth, with improvements saturating in a 4 deep channel for a buffer count of 2 and in a 8 deep channel for a count of 8. This number was independent of the type of buffering policy employed as well as the burst length being used.

Fig 5.45 shows the performance improvements achieved by using Source Buffering for different channel lengths and burst lengths. Fig 5.46 shows the performance improvements achieved by buffering at any AMB for an unbounded duration for different channel lengths and burst lengths. Fig 5.47 shows the performance improvements achieved by buffering at any AMB for a fixed duration, in this case an entire burst length, for different channel lengths and burst lengths. Fig 5.49 shows the performance improvements achieved by using the different buffering approaches for different channel lengths and burst lengths. Figure 5.50 shows the variation in contributors to average read latency as the number of buffers is changed for various channel depths for a burst length of 8 while figure 5.51 shows it for a burst length of 4. Figure 5.52 and 5.53 shows how the variation in the latency contributors for the different buffering policies used for burst length 4 and 8 systems respectivel.

As expected, configurations with longer channels benefitted the most from using buffering. In these systems, the north link is unavailable for longer durations because of the longer flight time for read data frames. Buffering allows transactions to move data out of the DRAM and holds it in the buffer till the earlier scheduled transaction is done transmitting. By de-coupling the data burst out of the DRAM from the data burst on the north link, buffering is able to schedule transactions to earlier DIMMs in the system as soon as the DRAM resources are available. As in the case of re-ordering, being able to schedule requests to earlier DIMMs contributes to the bulk of the latency reductions.

Buffering improves average read latency by reducing the queueing delay due to north link being unavailable and the transaction queue being unavailable (Figure 5.50 and 5.51). However, buffering does not completely eliminate the queueing delay associated with the



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

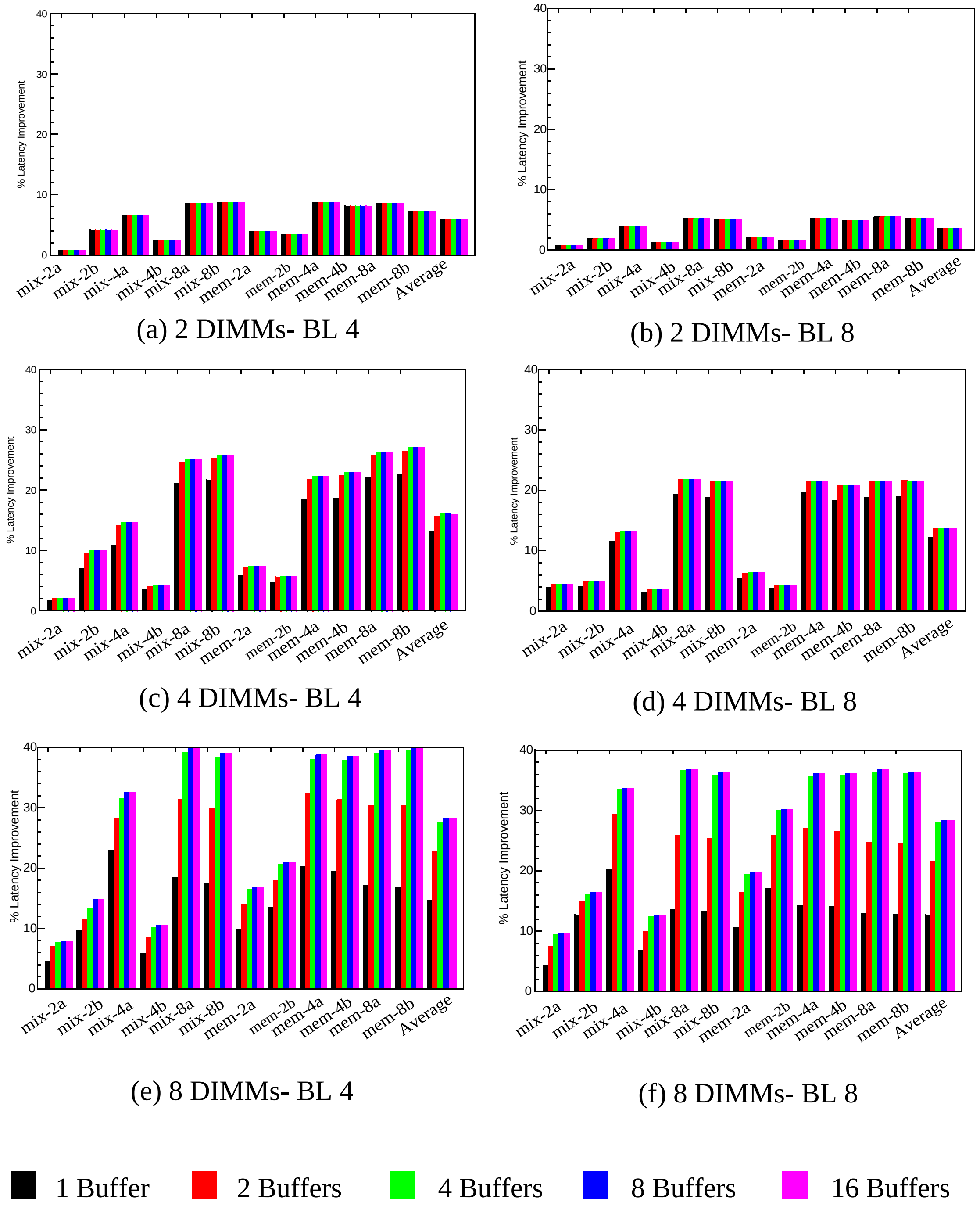
(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

(f) 8 DIMMs- BL 8

Figure 5.45. Latency Improvement by using Source Buffering . The graphs plot the percentage improvements for the average read latency on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with a bar for a different buffer count.



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

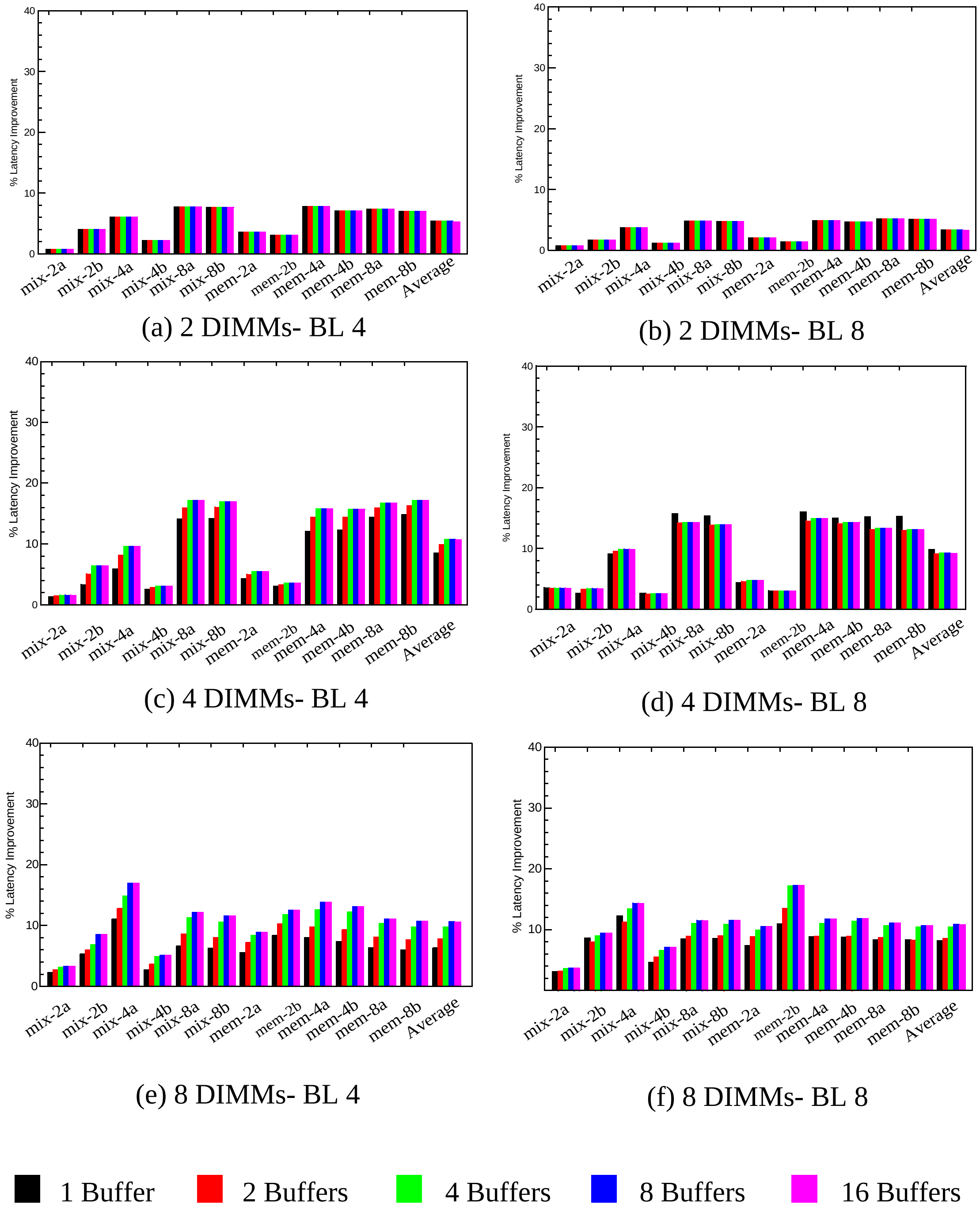
(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

(f) 8 DIMMs- BL 8

Figure 5.46. Latency Improvement by using buffer anywhere for variable duration. The graphs plot the percentage improvements for the average read latency on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with a bar for a different buffer count.



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

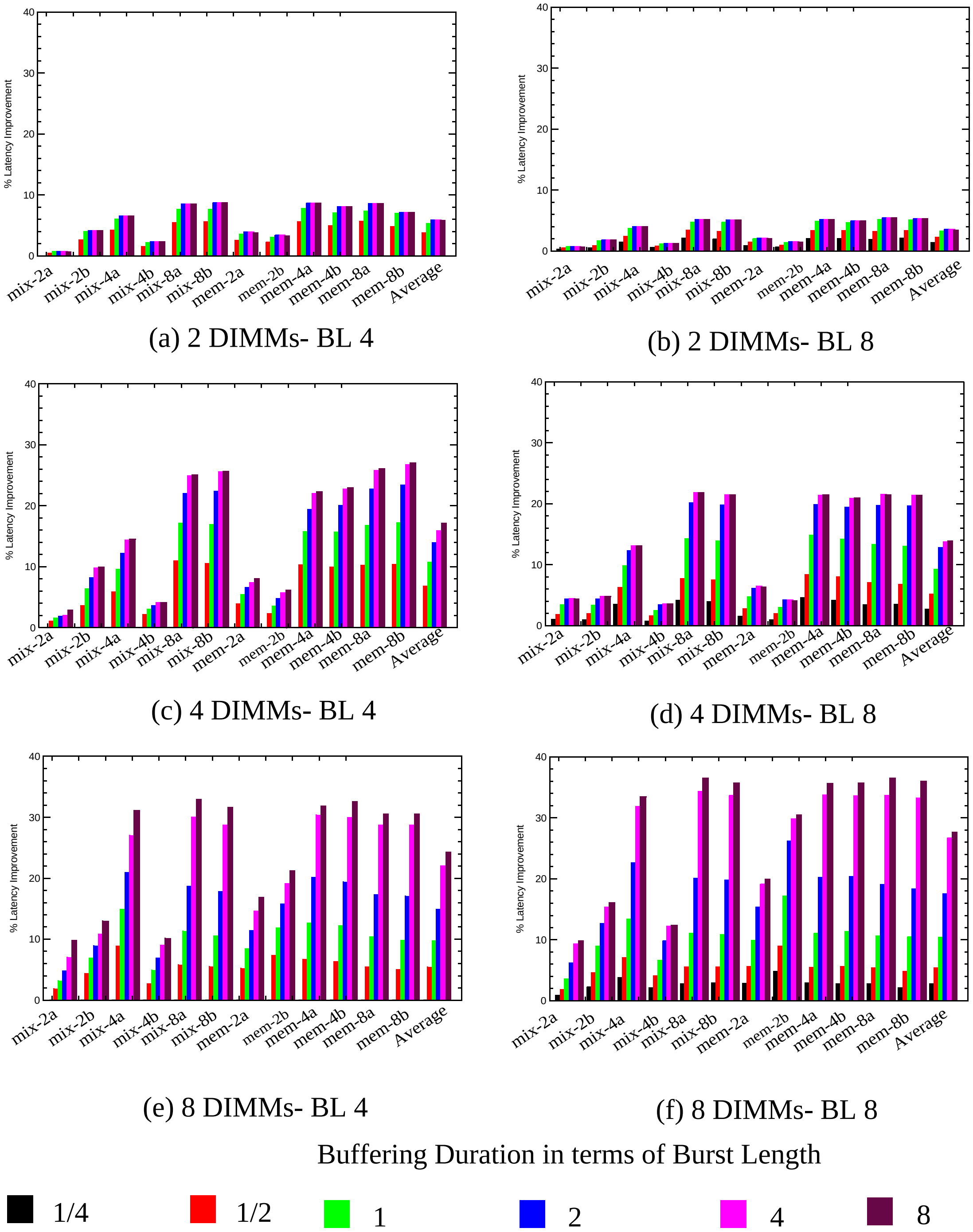
(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

(f) 8 DIMMs- BL 8

Figure 5.47. Latency Improvement by using buffer fixed anywhere - whole burst The graphs plot the percentage improvements for the average read latency on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with a bar for a different buffer count. **.**



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

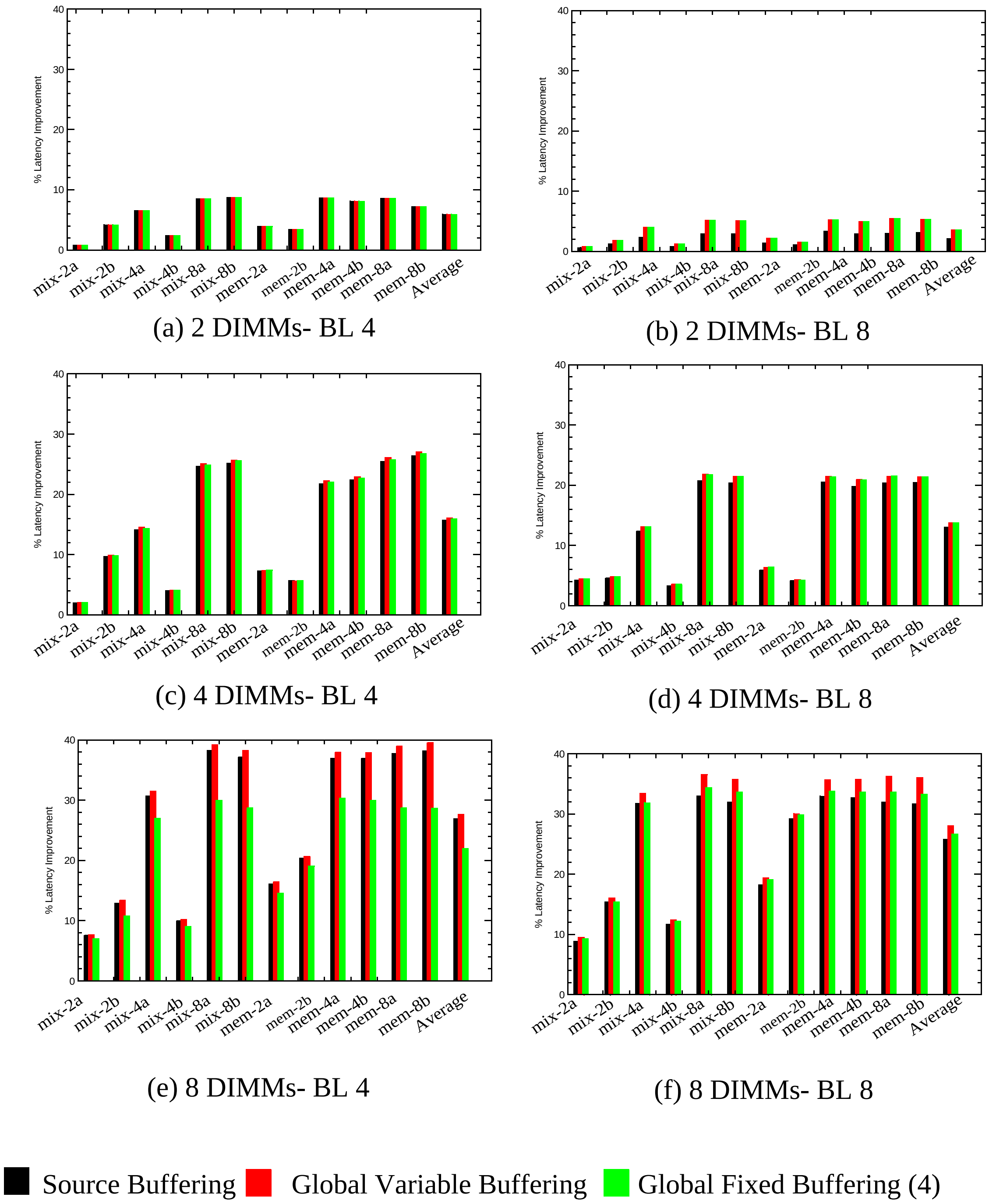
(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

(f) 8 DIMMs- BL 8

Figure 5.48. Latency Improvement by using different durations for Global Binary Buffering. The graphs plot the percentage improvements for the average read latency on the y-axis for the different applications at different channel depth. Each application’s data i grouped on the x-axis with a bar for a different buffer count.



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

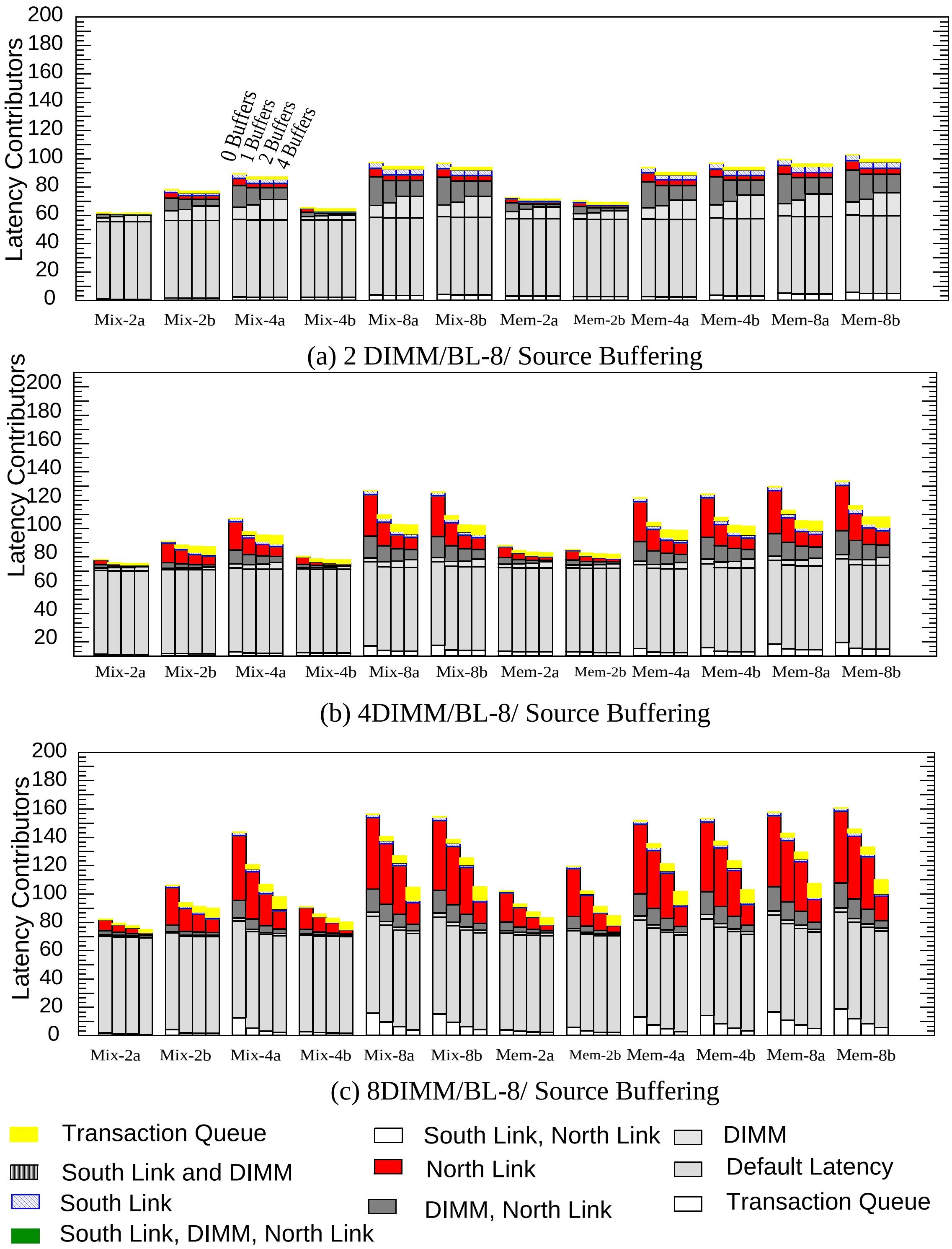
(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

(f) 8 DIMMs- BL 8

Figure 5.49. Latency Improvements by using different buffering schemes for a system using in-order return. The graphs plot the percentage improvements for the average read latency on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with a bar for a different buffering scheme. Each AMB can buffer 4 data bursts.

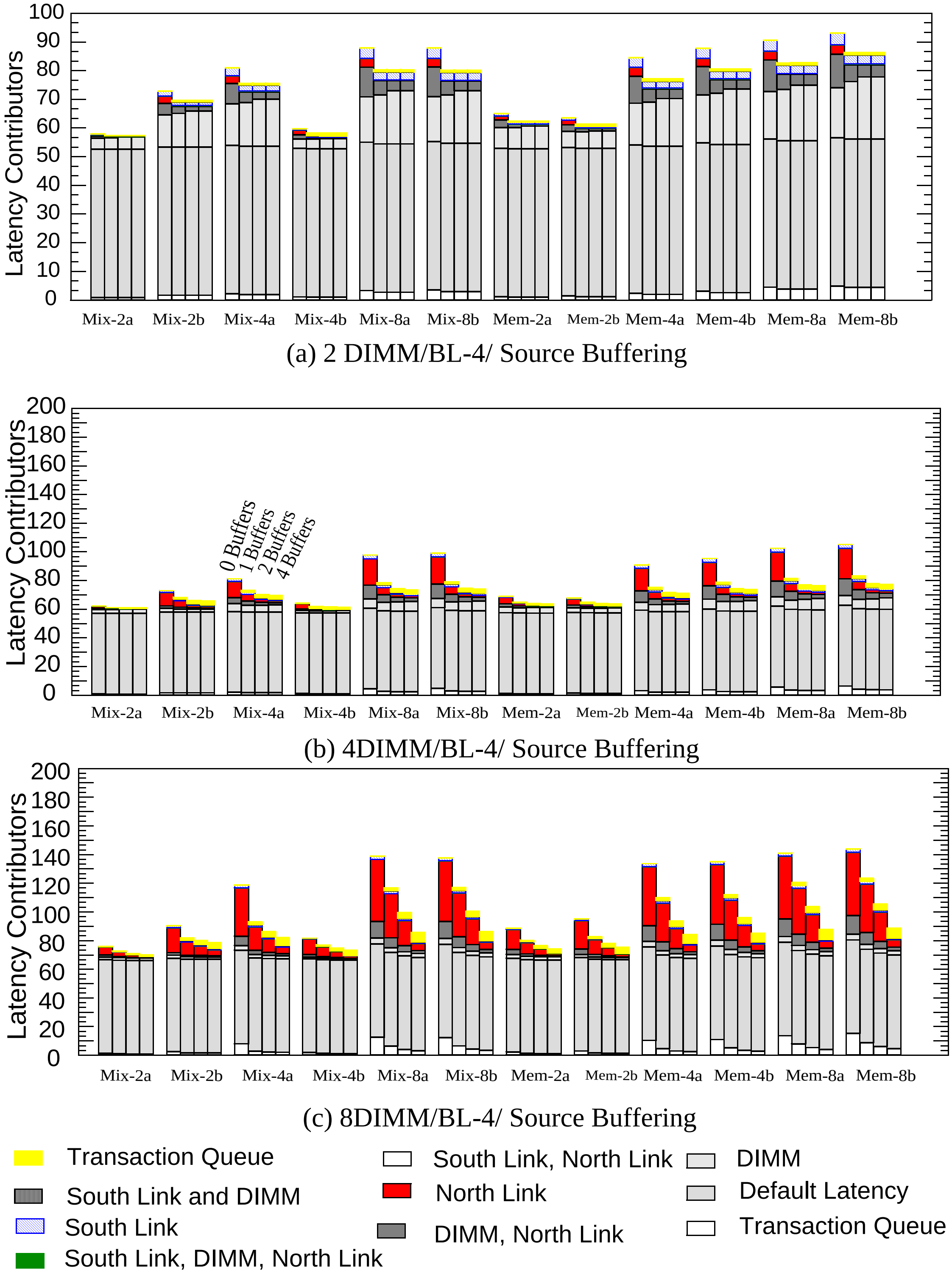


(a) 2 DIMM/BL-8/ Source Buffering

(b) 4DIMM/BL-8/ Source Buffering

(c) 8DIMM/BL-8/ Source Buffering

Figure 5.50. Impact of number of buffers on read latency contributors for various channel depths. The graphs are plotted for a single channel closed page FBD-DDR3 system which uses a burst length of 8 and Source Buffering. The y-axis plots the latency in ns, split into its various contributing causes for the various workload combinations which are shown on the x-axis.

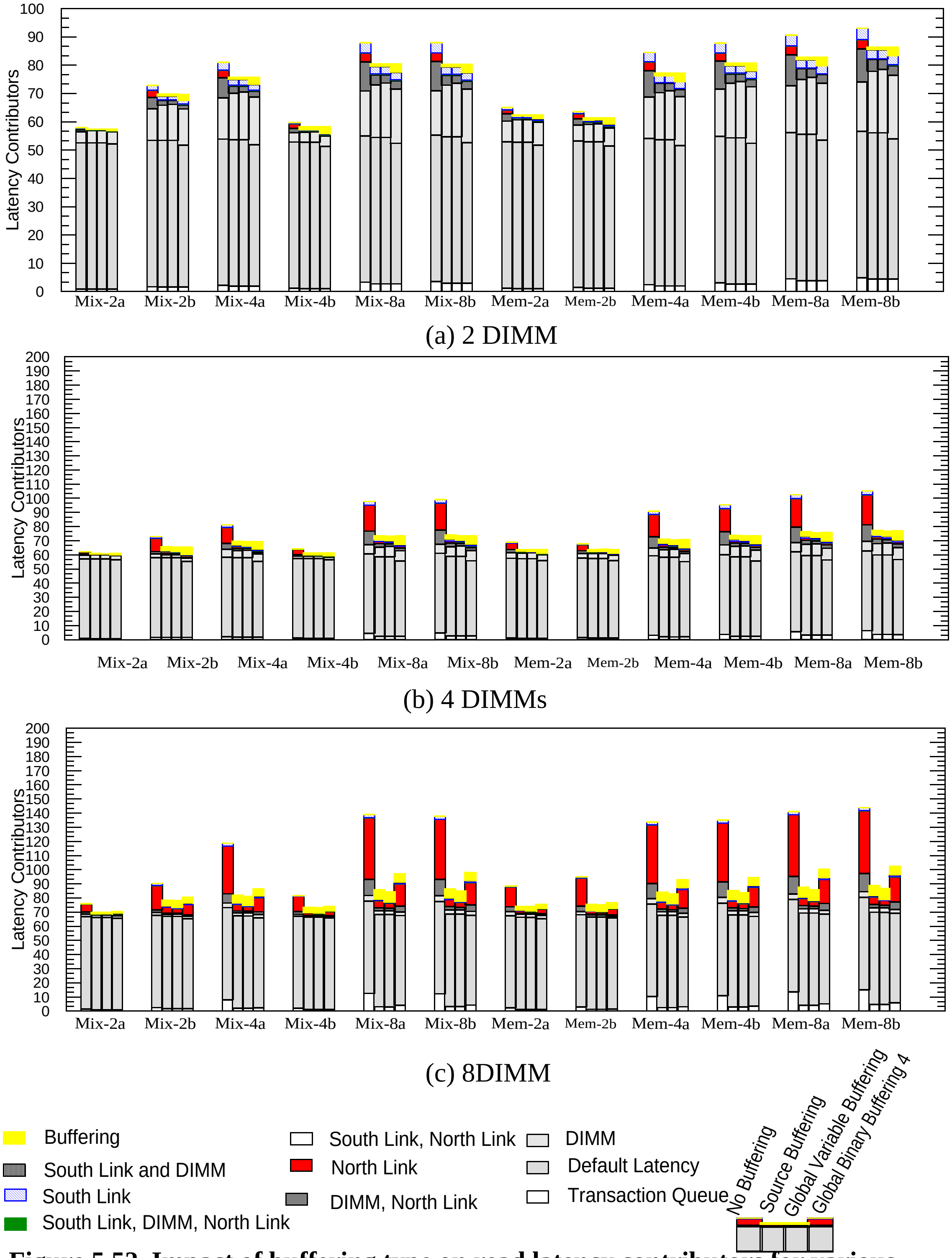


(a) 2 DIMM/BL-4/ Source Buffering

(b) 4DIMM/BL-4/ Source Buffering

(c) 8DIMM/BL-4/ Source Buffering

Figure 5.51. Impact of number of buffers on read latency contributors for various channel depths. The graphs are plotted for a single channel closed page FBD-DDR3 system which uses a burst length of 4 and Source Buffering. The y-axis plots the latency in ns, split into its various contributing causes for the various workload combinations which are shown on the x-axis.

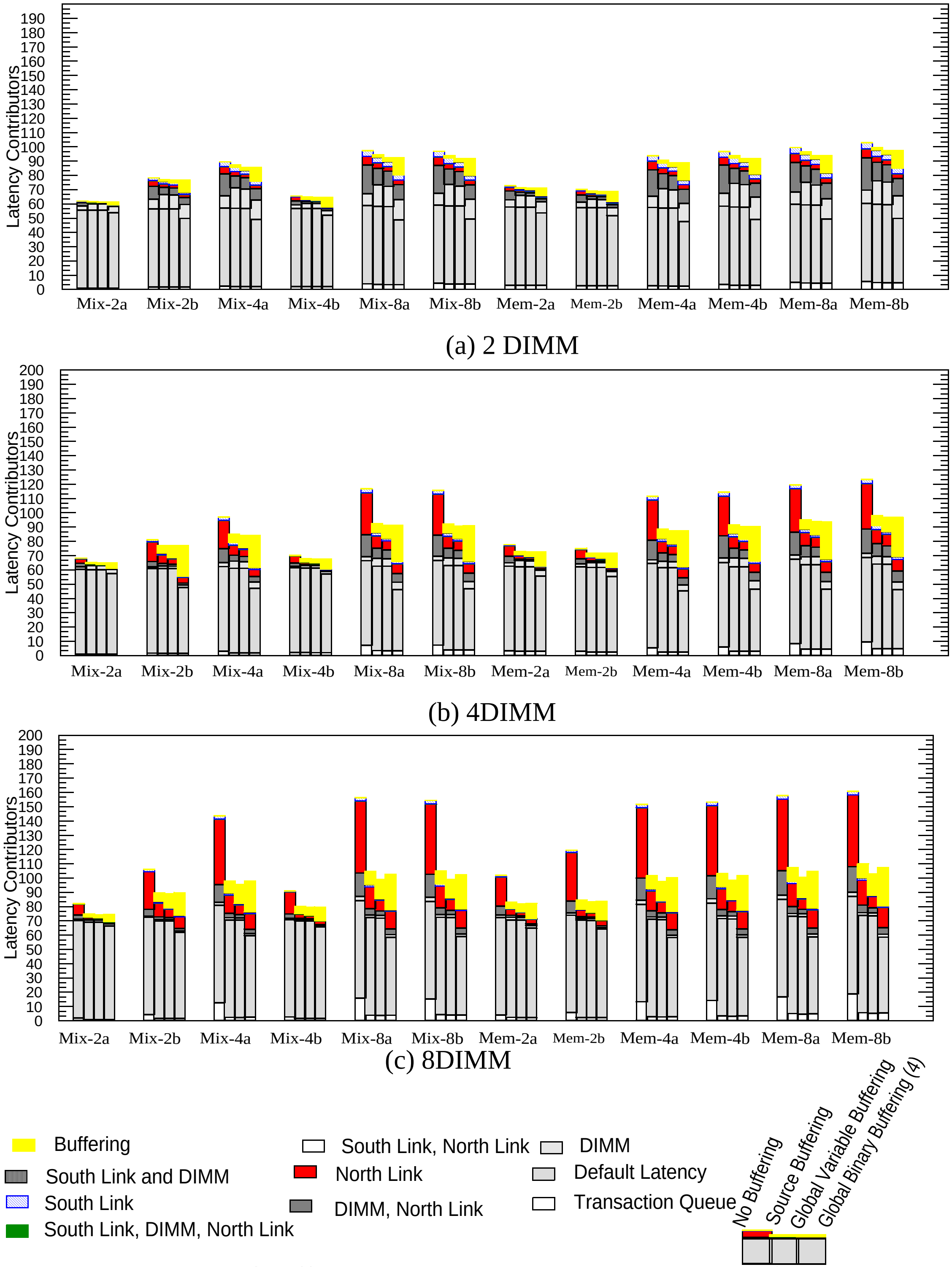


(a) 2 DIMM

(b) 4 DIMMs

(c) 8DIMM

Figure 5.52. Impact of buffering type on read latency contributors for various channel depths for burst length 4 systems. The graphs are plotted for a single channel closed page FBD-DDR3 system which uses a burst length of 4 and 4 buffers. The y-axis plots the latency in ns, split into its various contributing causes for the various workload combinations which are shown on the x-axis.



(a) 2 DIMM

(b) 4DIMM

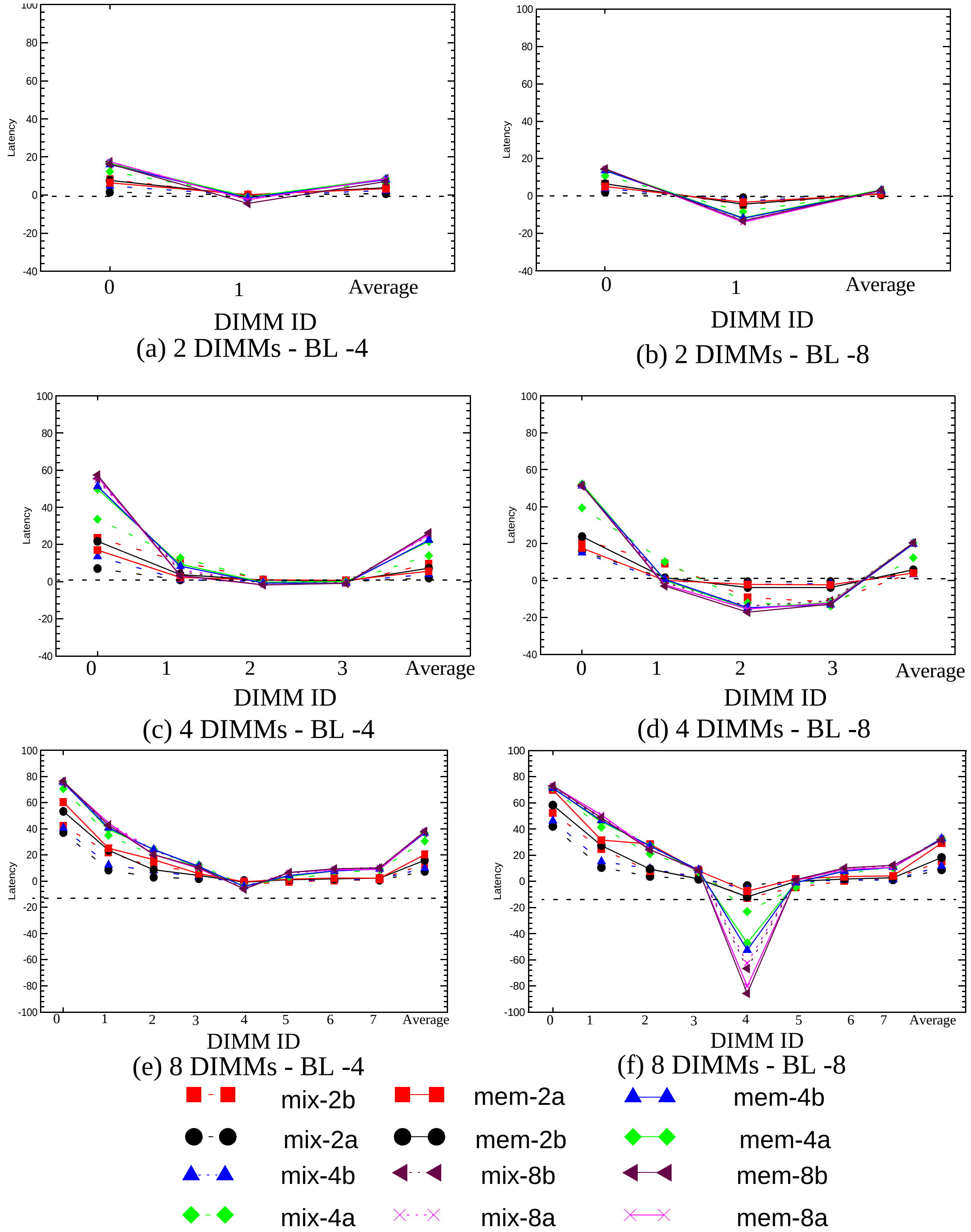
(c) 8DIMM

Figure 5.53. Impact of buffering type on read latency contributors for various channel depths for burst length 8 systems. The graphs are plotted for a single channel closed page FBD-DDR3 system which uses a burst length of 8 and 4 buffers. The y-axis plots the latency in ns, split into its various contributing causes for the various workload combinations which are shown on the x-axis.

unavailability of the north link in burst length 4 systems (Figure 5.51) and is less effective in the case of burst length 8 systems as well (Figure 5.50). This is because buffering does not utilize idle gaps in the north link. Instead, buffering delays the point at which the north link is used and thus replaces some of the north link queueing delay with a buffering delay. Hence, we see in figures 5.50 and 5.51 that by using buffering the north link queueing delay drops, but the buffering overheads increase. Despite the cost of buffering, we find that the ability to schedule requests to

Figures 5.54, 5.55 and 5.56 plot the improvements in per DIMM average read latency for a system using Buffering at Source, Global Variable Buffering and Global Binary Buffering with buffer duration set to one entire burst length. In the case of Source Buffering and Global Variable Buffering (Figure5.54 and 5.55), we see that buffering reduces the latency of transactions to the DIMM closest to the memory controller, but typically increases the latency of transactions to DIMMs which are further from the memory controller. The degradation in latency is more pronounced for a system with burst length 8 (Figure5.54 b, d and f and 5.55 b, d and f) and for systems using Source Buffering. Further, the worst case latency degradation increases with the number of DIMMs in the channel.

Examining the contributors to latency for transactions to a given DIMM in figures we see that buffering significantly reduces the queueing delay experienced by transactions to earlier DIMMs in the system, but simultaneously increases the queueing delay due to the north link being unavailable experienced by transactions to DIMMs which are further from the memory controller. Buffering allows the memory controller to schedule transactions to the first DIMM in the system without having to wait for transactions to other DIMMs to complete. By doing this, the memory controller increases the utilization of the northbound



(a) 2 DIMMs - BL -4

(b) 2 DIMMs - BL -8

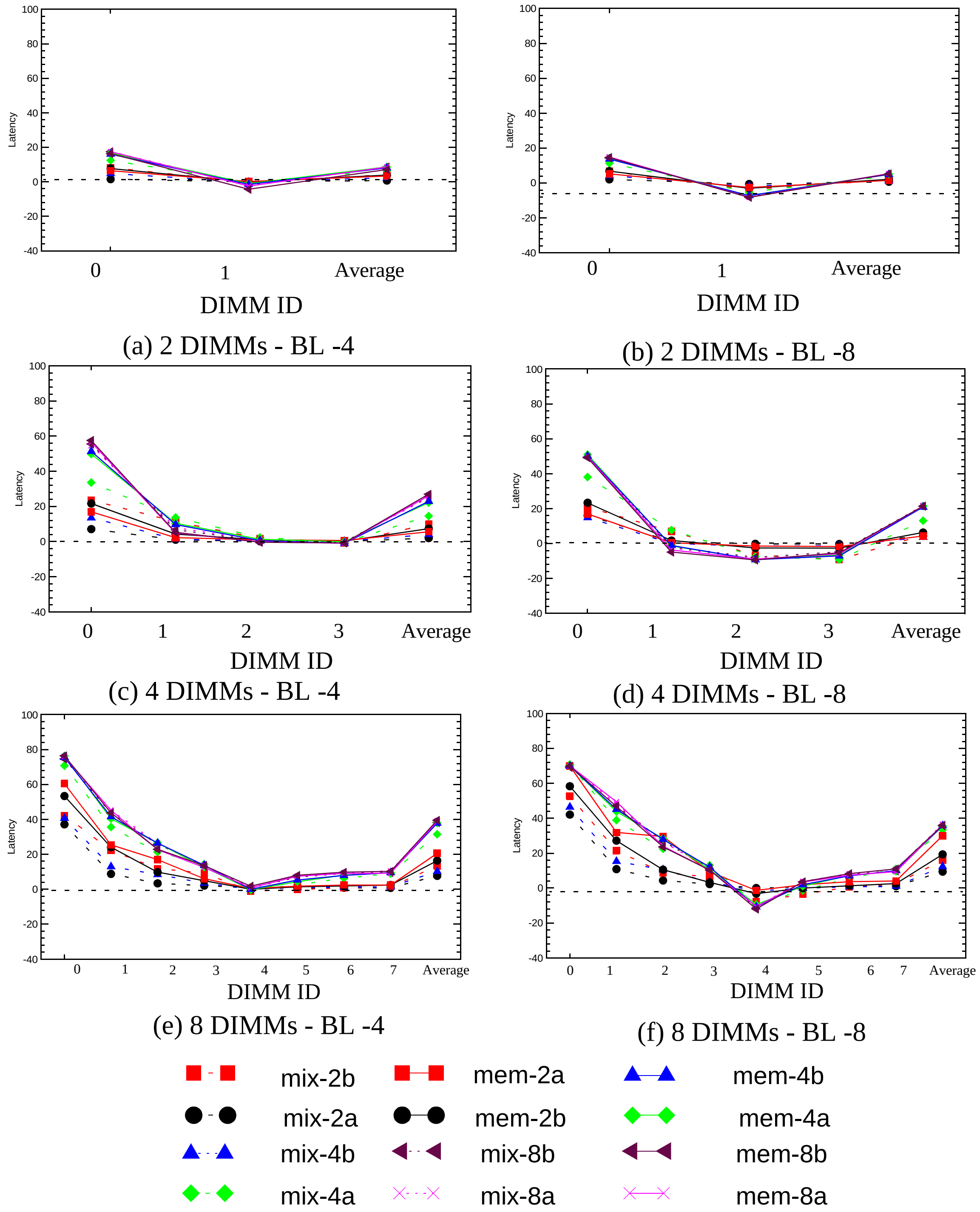
(c) 4 DIMMs - BL -4

(d) 4 DIMMs - BL -8

(e) 8 DIMMs - BL -4

(f) 8 DIMMs - BL -8

Figure 5.54. Improvements in Average Latency of a transaction to a given DIMM using Buffering at Source. The graphs in the figure shows the variation in read latency for a transaction addressed to a particular DIMM in the system. On the y-axis is plotted the percentage improvement in latency for each DIMM's transactions. The various DIMMs are plotted on the x-axis, with the numbering starting with the DIMM closest to the memory controller. Note the last point is the overall average read latency improvement experienced. Graph f has a different y-axis.



(a) 2 DIMMs - BL -4

(b) 2 DIMMs - BL -8

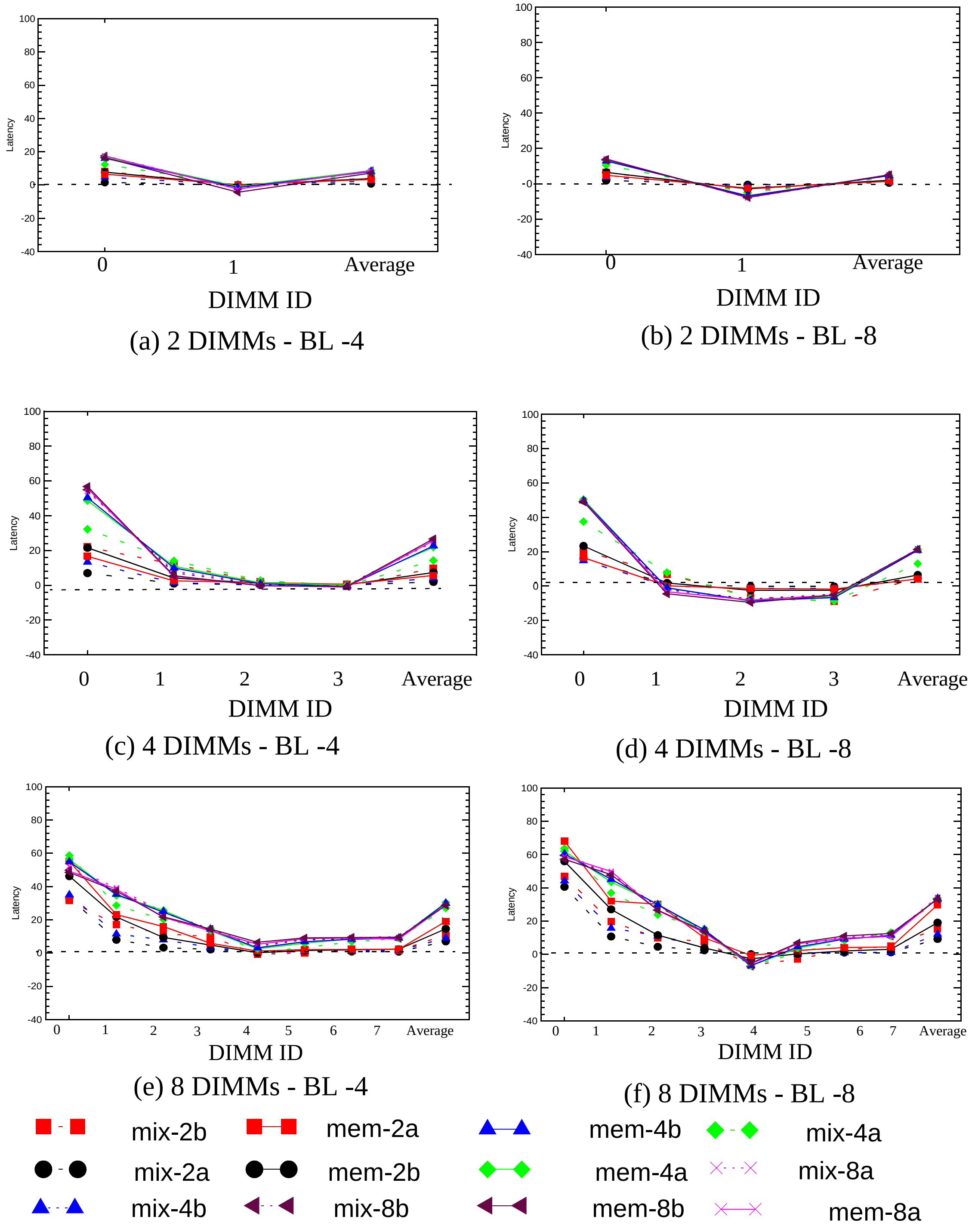
(c) 4 DIMMs - BL -4

(d) 4 DIMMs - BL -8

(e) 8 DIMMs - BL -4

(f) 8 DIMMs - BL -8

Figure 5.55. Improvements in Average Latency of a transaction to a given DIMM using Global Variable Buffering. The graphs in the figure shows the variation in read latency for a transaction addressed to a particular DIMM in the system. On the y-axis is plotted the percentage improvement in latency for each DIMM's transactions. The various DIMMs are plotted on the x-axis, with the numbering starting with the DIMM closest to the memory controller. Note the last point is the overall average read latency improvement experienced. Graph f has a different y-axis.



(a) 2 DIMMs - BL -4

(b) 2 DIMMs - BL -8

(c) 4 DIMMs - BL -4

(d) 4 DIMMs - BL -8

(e) 8 DIMMs - BL -4

(f) 8 DIMMs - BL -8

Figure 5.56. Improvements in Average Latency of a transaction to a given DIMM using Global Binary Buffering. The graphs in the figure shows the variation in read latency for a transaction addressed to a particular DIMM in the system. On the y-axis is plotted the percentage improvement in latency for each DIMM's transactions. The various DIMMs are plotted on the x-axis, with the numbering starting with the DIMM closest to the memory controller. Note the last point is the overall average read latency improvement experienced. Graph f has a different y-axis.

bus for transactions to this particular DIMM. Due to the latter phenomenon, transactions to the next couple of DIMMs have to wait longer for bursts to earlier DIMMs to complete. Transactions that are scheduled to further DIMMs (as in the case of DIMM 5,6,7,8 in a 8 DIMM deep channel) are not affected, because of their longer round-trip latency.

Global variable buffering is able to alleviate some of this increased queueing delay by allowing returning read data to be buffered at multiple points. This allows the memory controller to buffer the returning read data frames at any DIMM if the link immediately ahead of it is unavailable. Hence, in a system using Global Variable Buffering, the read data bursts to DIMMs which are sent to DIMMs further down the channel, are able to be scheduled without having to wait for the read data bursts from DIMM 1 to complete. Hence, we see by comparing figures 5.54 f and 5.55 f, that transactions to DIMM 4 experiences far less latency degradation when the system uses Global Variable Buffering then when the system uses Global Source Buffering.

The ability to buffer at multiple DIMMS useful in the case of memory intensive workloads which have larger bandwidth demands and can result in an overall 5% higher latency reduction for these applications by using Global Variable Buffering as compared to Global Source Buffering (fig 5.49). These reductions are larger in a burst length 8 system because of the longer data bus use per transaction. Figures 5.52 and 5.53 which show the variation in contributors to read latency with different buffering policies, establish that the queueing delay reductions for Global Variable Buffering is higher than for Global Source Buffering.

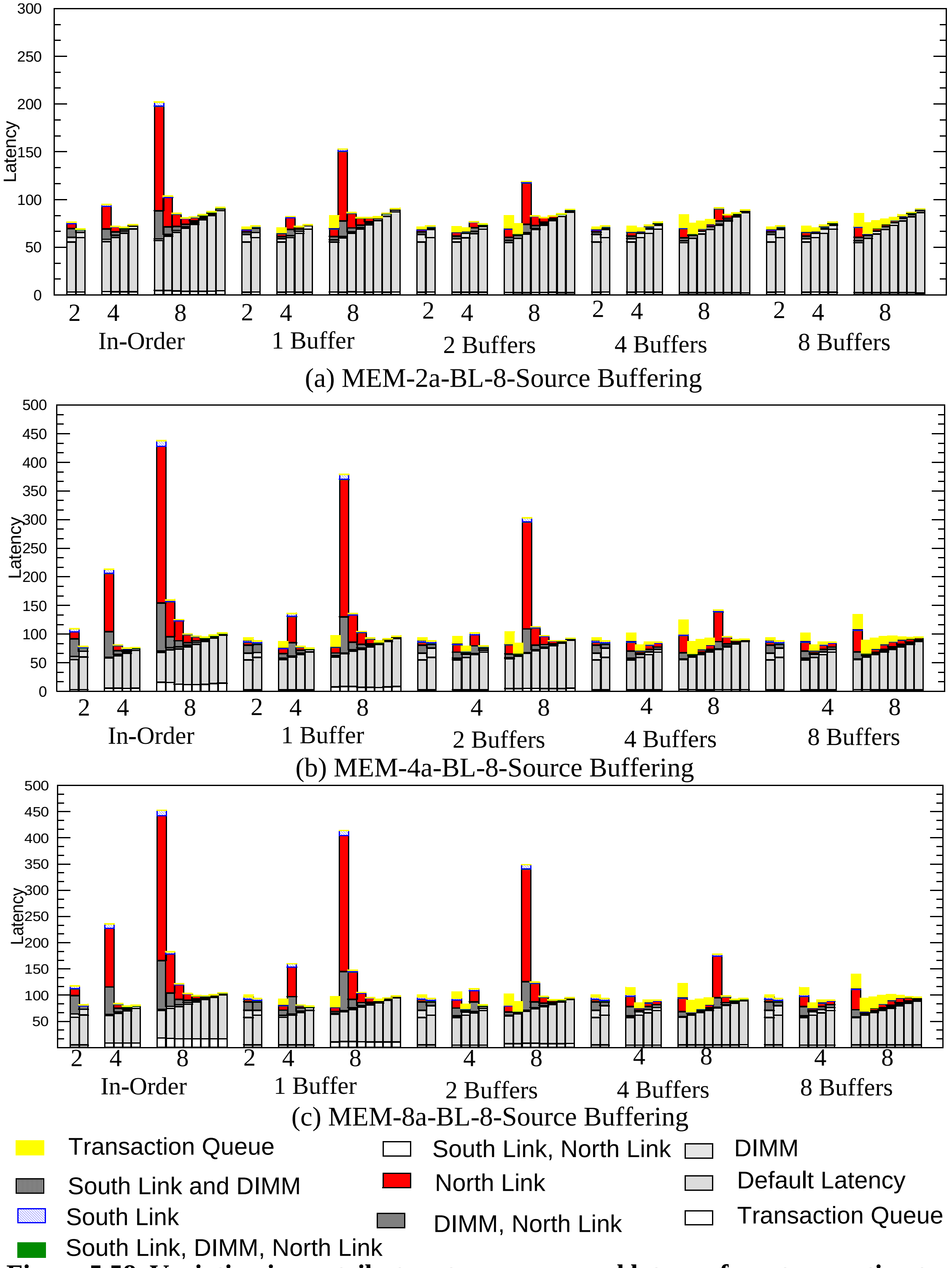
The reductions in latency due to the use of Global Binary Buffering increase with the buffering duration permitted (fig 5.48). The buffering gains taper out after a buffer duration of 4 times the burst duration is used. This behavior suggests that it is more likely for transactions to buffer for longer durations than shorter durations. Systems with 8 DIMM deep channels continue to benefit when the buffering duration in set to 8 times a burst duration.

Figure 5.49 shows that the three buffering schemes achieve comparable performance for systems with 4 or less DIMMs. For a channel with 8 DIMMs we find that global binary buffering yields the lowest improvements while global variable buffering did the best.

Increasing the number of buffers on each AMB typically improves the average read latency till a certain buffer count after which no additional benefits are seen. The buffer count at which the gains taper off is a function of the channel length, with 2 DIMM deep systems seeing the maximum possible benefits with one buffer, 4 DIMM deep system seeing maximum possible benefits with two buffers and 8 DIMM deep systems 4 buffers (Figures 5.45, 5.46 and 5.47). In the case of burst length 4 systems, the latency reductions stop when the north link queueing delay has been eliminated and a part of it has been replaced with the buffering delay (figure 5.51 (a, b)). In the case of burst length 8 systems, no further reductions in queueing delay are seen by adding more buffers (figure 5.50). By examining figure 5.58 and, we see how the contributors to average read latency experienced by a transaction to a given DIMM changes as more DIMMs are added in the system.

5.3.2.1 Bandwidth Improvements

Fig 5.59 shows the performance improvements achieved by using Source Buffering for different channel lengths and burst lengths. Fig 5.60 shows the performance improvements achieved by buffering at any AMB for an unbounded duration for different channel lengths and burst length. Fig 5.61 shows the performance improvements achieved by buffering at any AMB for a fixed duration which is equal to an entire burst length for different

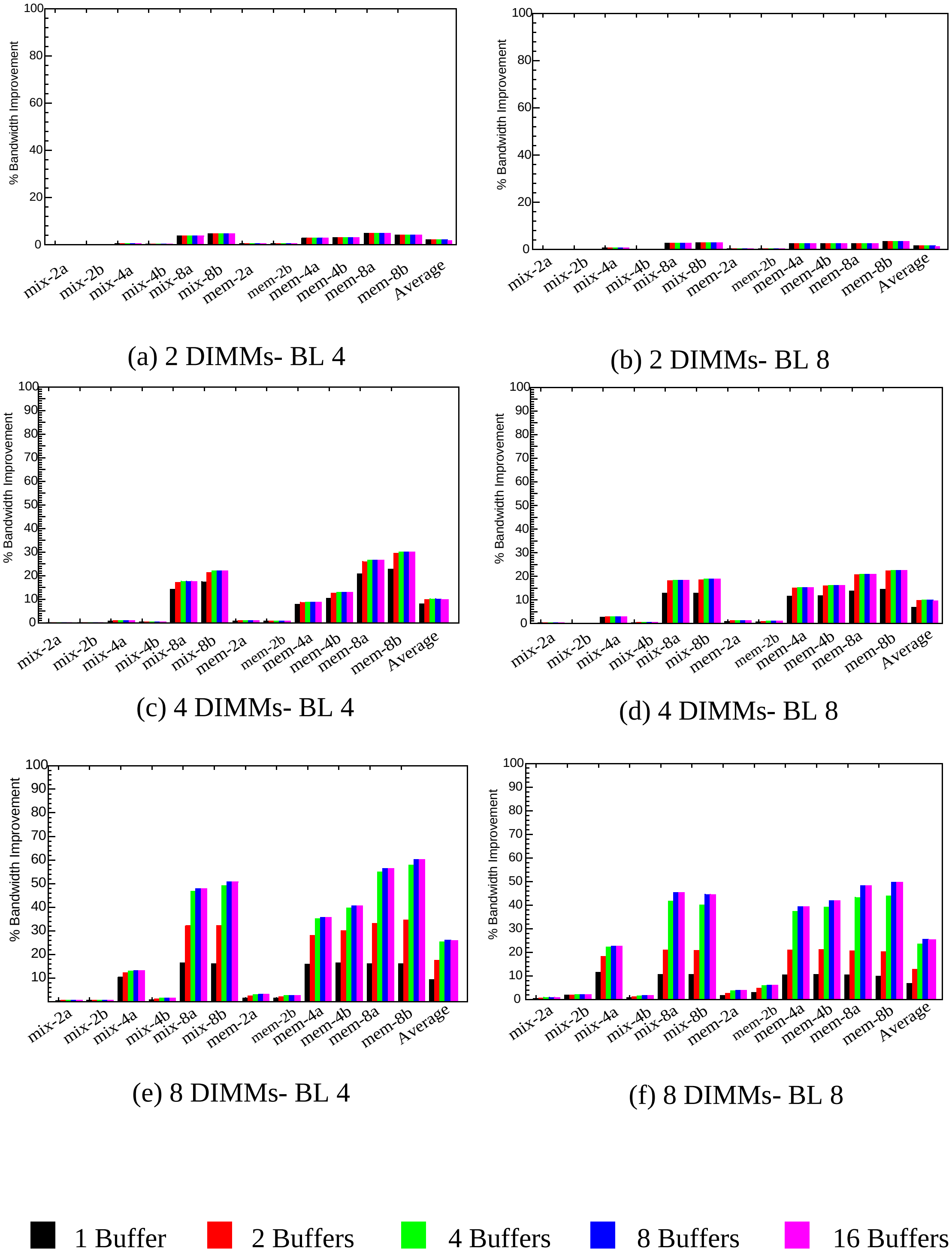


(a) MEM-2a-BL-8-Source Buffering

(b) MEM-4a-BL-8-Source Buffering

(c) MEM-8a-BL-8-Source Buffering

Figure 5.58. Variation in contributors to average read latency for a transaction to a given DIMM with number of buffers. The graphs plot the variation for a system with a burst length of 8 using Source Buffering. The x-axis is grouped by configurations with a given number of DIMMs in the system. Each such group is part of a larger group of data for 2, 4 and 8 DIMM deep channels. The larger groups are data for a particular



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

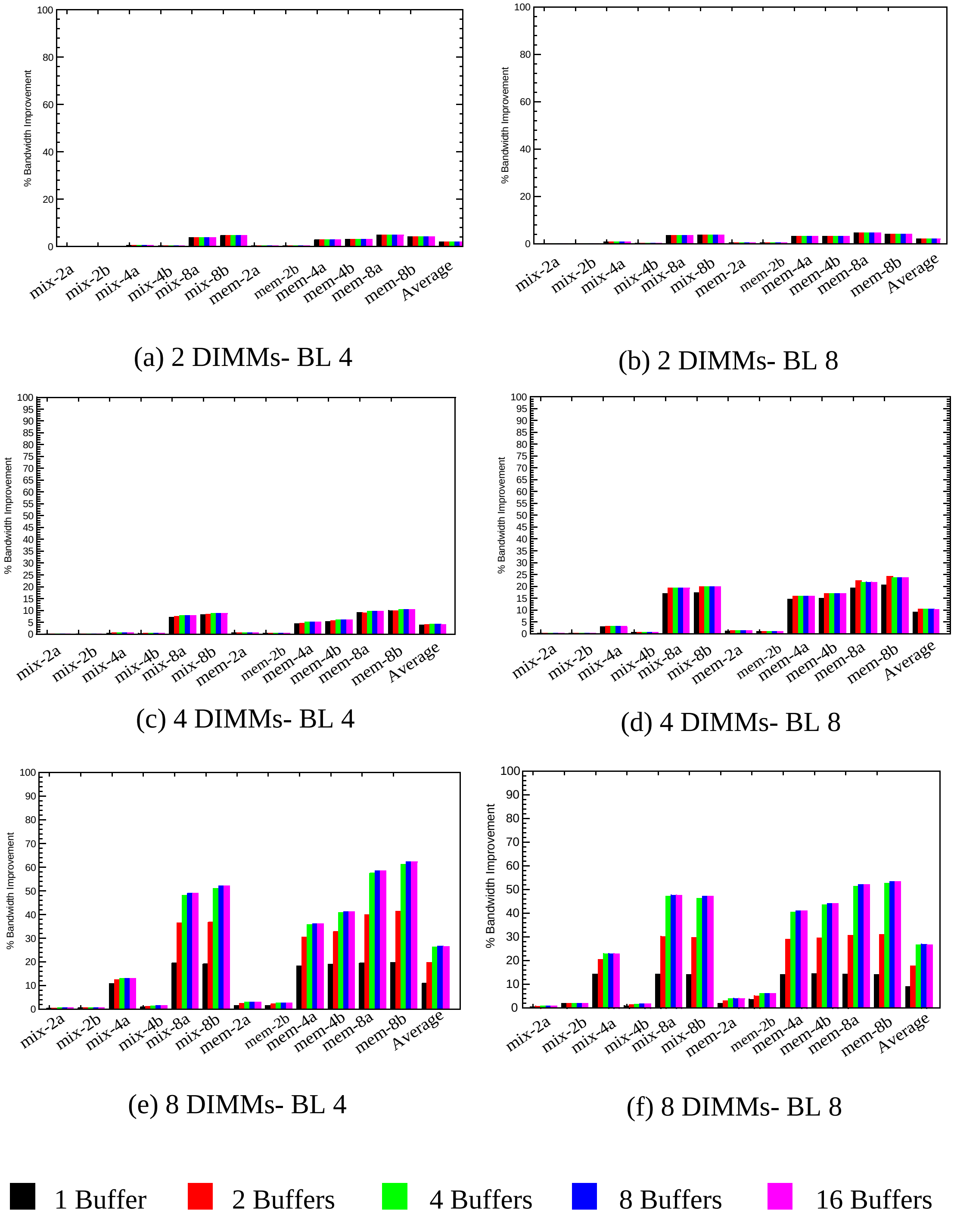
(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

(f) 8 DIMMs- BL 8

Figure 5.59. Bandwidth Improvement by using Source Buffering . The graphs plot the percentage improvements in bandwidth on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with a bar for a different buffer count.



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

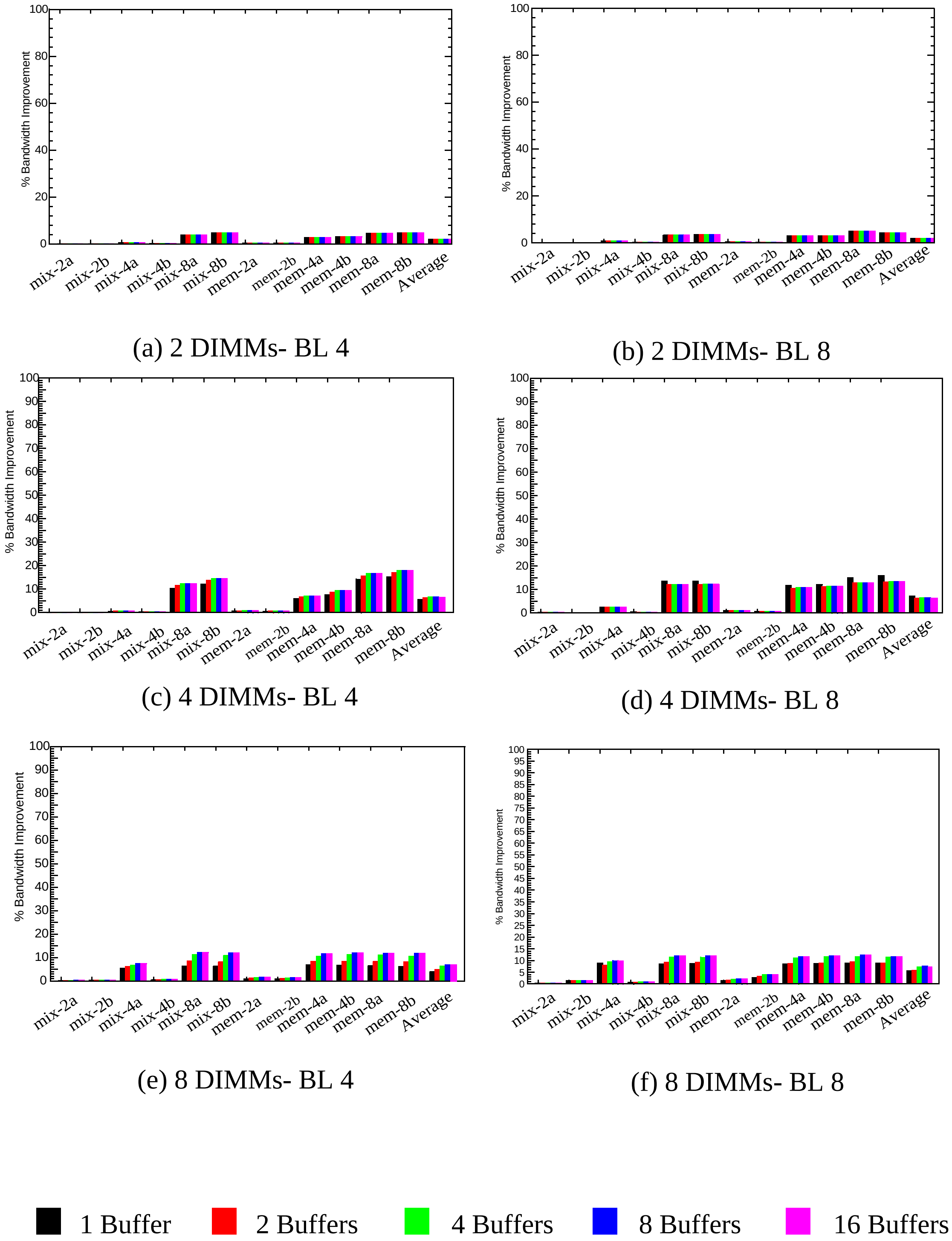
(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

(f) 8 DIMMs- BL 8

Figure 5.60. Bandwidth Improvement by using Global Variable Buffering. The graphs plot the percentage improvements for the average read latency on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with a bar for a different buffer count.



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

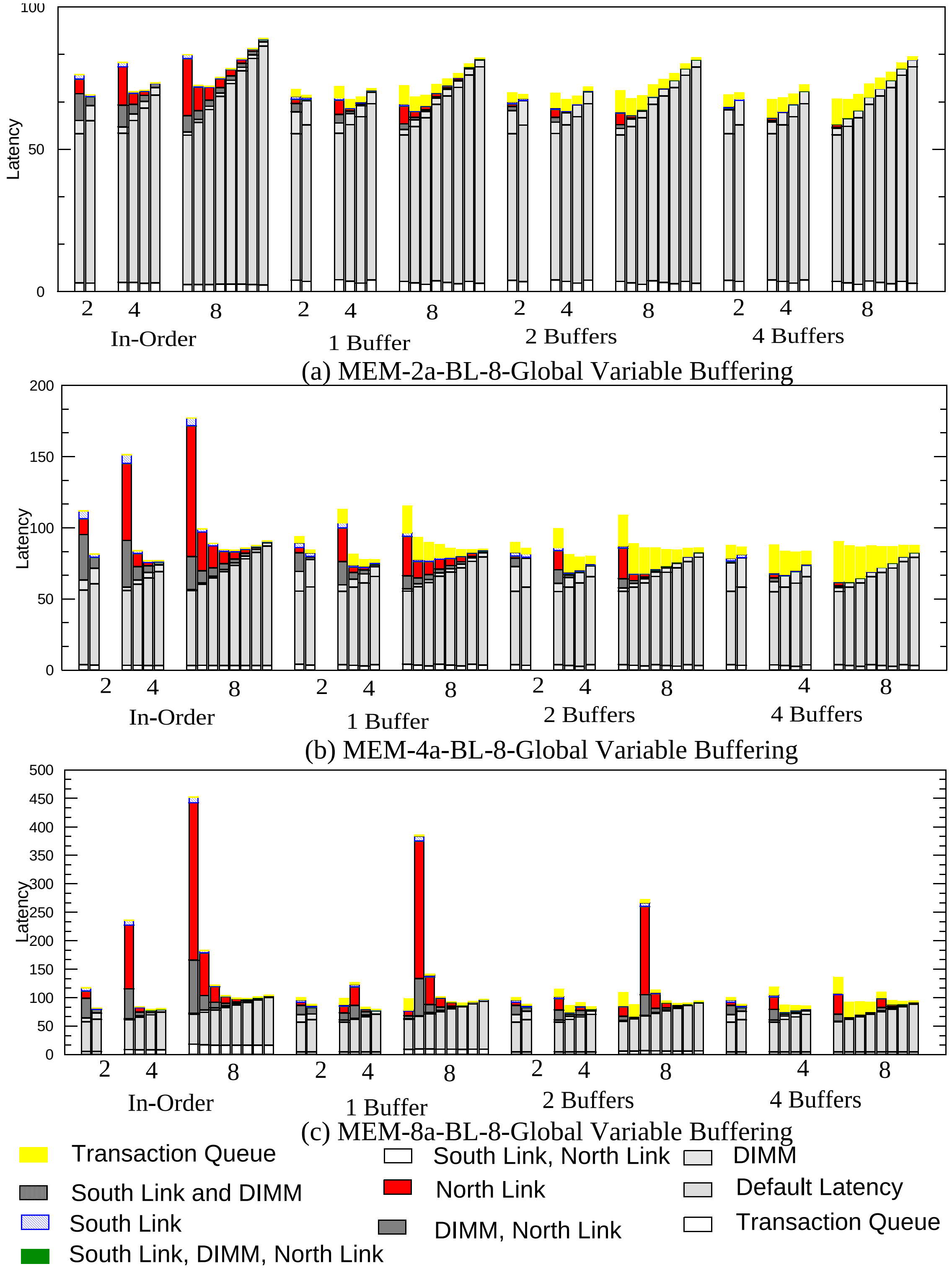
(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

(f) 8 DIMMs- BL 8

Figure 5.61. Bandwidth Improvement by using buffer fixed anywhere - whole. The graphs plot the percentage improvements for the average read latency on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with a bar for a different buffer count.



(a) MEM-2a-BL-8-Global Variable Buffering

(b) MEM-4a-BL-8-Global Variable Buffering

(c) MEM-8a-BL-8-Global Variable Buffering

Figure 5.57. Variation in contributors to average read latency for a transaction to a given DIMM with number of buffers. The graphs plot the variation for a system with burst length of 8 using Source Buffering. The x-axis is grouped by configurations with given number of DIMMs in the system. Each such group is part of a larger group of data fo 2, 4 and 8 DIMM deep channels. The larger groups are data for a particular buffer size.

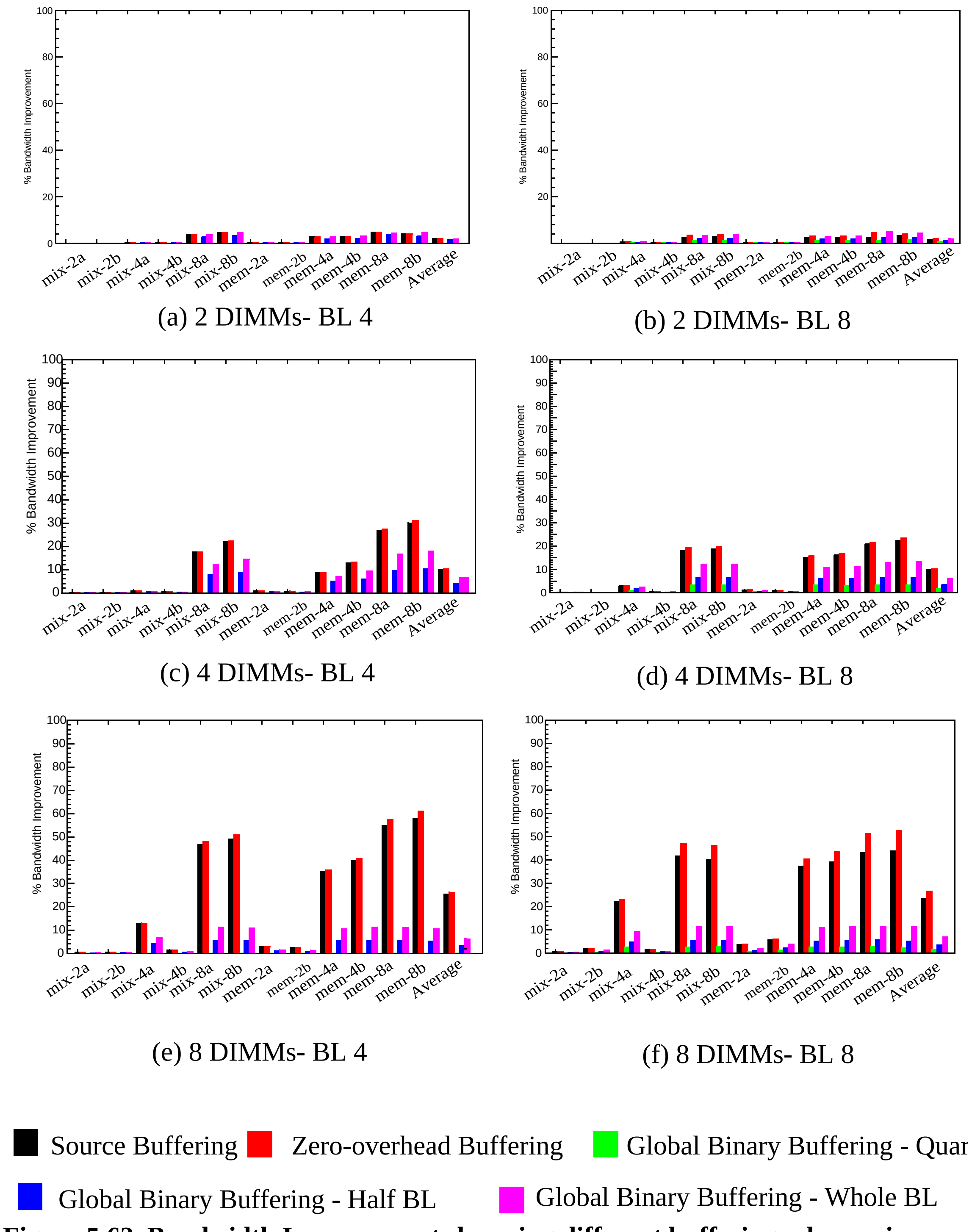
channel lengths and burst lengths. Fig 5.68 shows the bandwidth improvements achieved by using the different buffering approaches for different channel lengths and burst lengths for a AMB with 4 buffers.

As in the case of the latency improvements, we see that longer channels had the largest bandwidth improvements, with 8 deep channels benefitting by an average of 25% as compared to 5 and 10% for channel lengths of 2 and 4 DIMMs respectively. The benefits were more pronounced for the memory intensive workloads and the mix workloads with more threads, because they were able to utilize buffering to move more read data back to the memory controller. Since buffering does not schedule read transactions to utilize the gaps in the channel, the benefits of using buffering are less than that of using re-ordering.

We once again found that global variable buffering had better bandwidth improvements than buffering at source, with larger improvements seen for systems with longer channel depths, larger burst lengths and for workloads with higher bandwidth requirements.

5.4. Re-ordering and Buffering

We studied the impact on performance characteristics of a system that permits re-ordering of data returns as well as buffering. In general we found that the latency improved by an additional 5-8% for a 2 DIMM system, 5-10% for a 4 DIMM system and 5-!0% for an 8 DIMM system over a system which only permitted re-ordering of return data. The bandwidth improvements were on the order of 5% for a system with a burst length of 8 and around 2% for a system with a burst length of 4. Unlike a system which allowed data to return in-order we found that the buffering policy employed had no impact on the overall



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

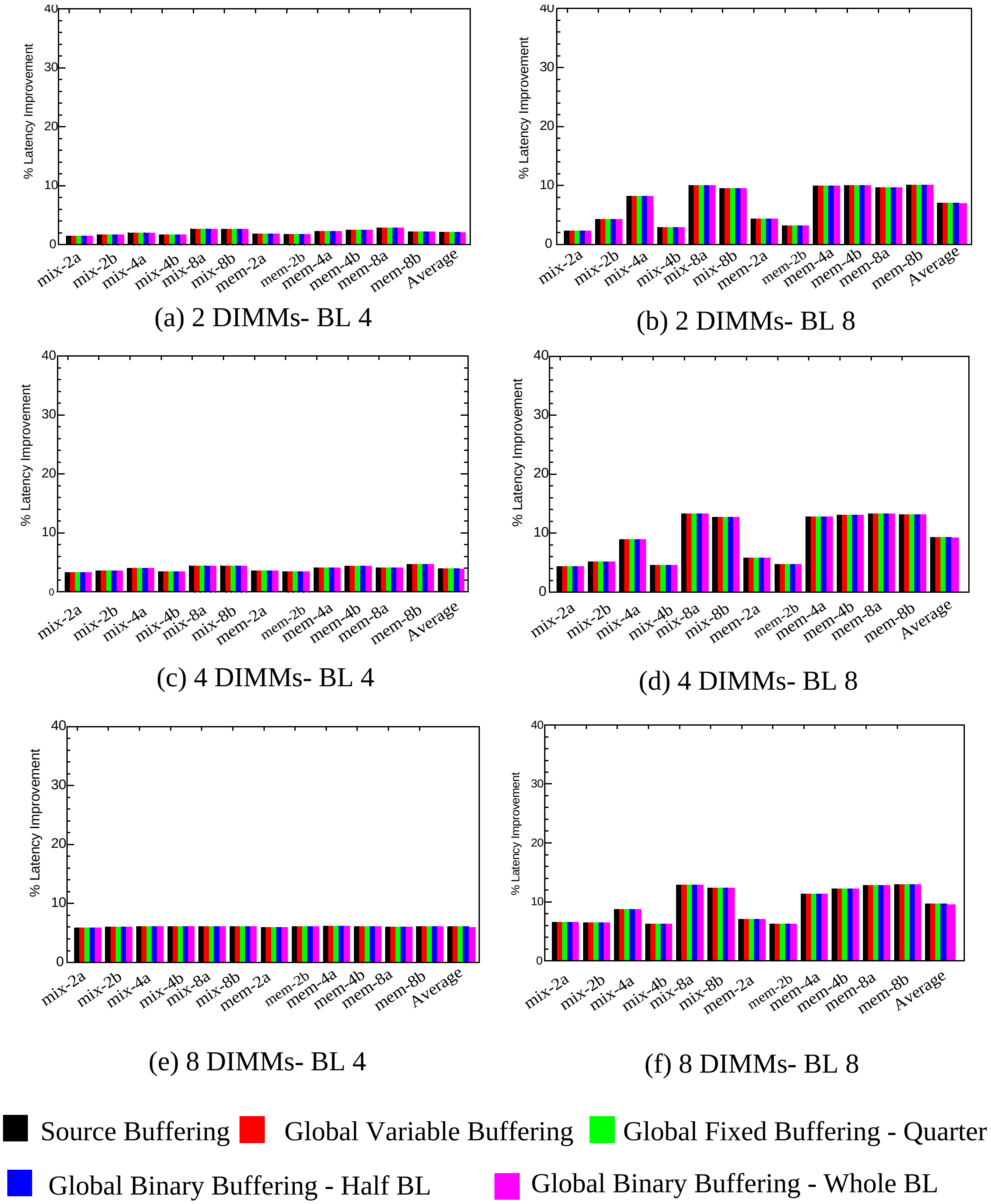
(f) 8 DIMMs- BL 8

Figure 5.62. Bandwidth Improvements by using different buffering schemes in in-order return system. The graphs plot the percentage improvements for the average read latency on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with a bar for a different buffering scheme.

performance improvements, while the number of buffers used made a difference mainly in a burst length 8 system.

Figure 5.63 shows the latency improvements achieved by using the different buffering approaches for different channel lengths and burst lengths for a AMB with 4 buffers. Figure5.64 shows the performance improvements achieved by using Source Buffering for different channel lengths and burst lengths. Figure and 5.66 5.67 shows the improvements in per-DIMM latency improvements by using Source Buffering. Figure5.69 shows the bandwidth improvements achieved by using the different buffering approaches for different channel lengths and burst lengths for a AMB with 4 buffers. Figure5.64 shows the bandwidth improvements achieved by using Source Buffering for different channel lengths and burst lengths.

Using buffering with re-ordering resulted in a further reduction in latency by eliminating any queueing delay associated with only the north link being available and by substantially reducing any queueing delay (fig 5.67) due to the north link and the DIMM being unavailable (figure 5.66). In the case of burst length 4 systems (figure 5.66) we see a single buffer was able to eliminate this remaining delay, while in a burst length 8 system more buffers were required (fig 5.67). In nearly all cases only a small fraction of the queueing delay was not translated into buffering overheads. The bulk of the overheads were converted into buffering overheads. The advantages of using buffering with re-ordering largely arise from being able to de-couple the wait for the north link from the wait for the DRAM. Hence, the system is able to more fully utilize the available DRAM-level parallelism and be only limited by the total available bandwidth.



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

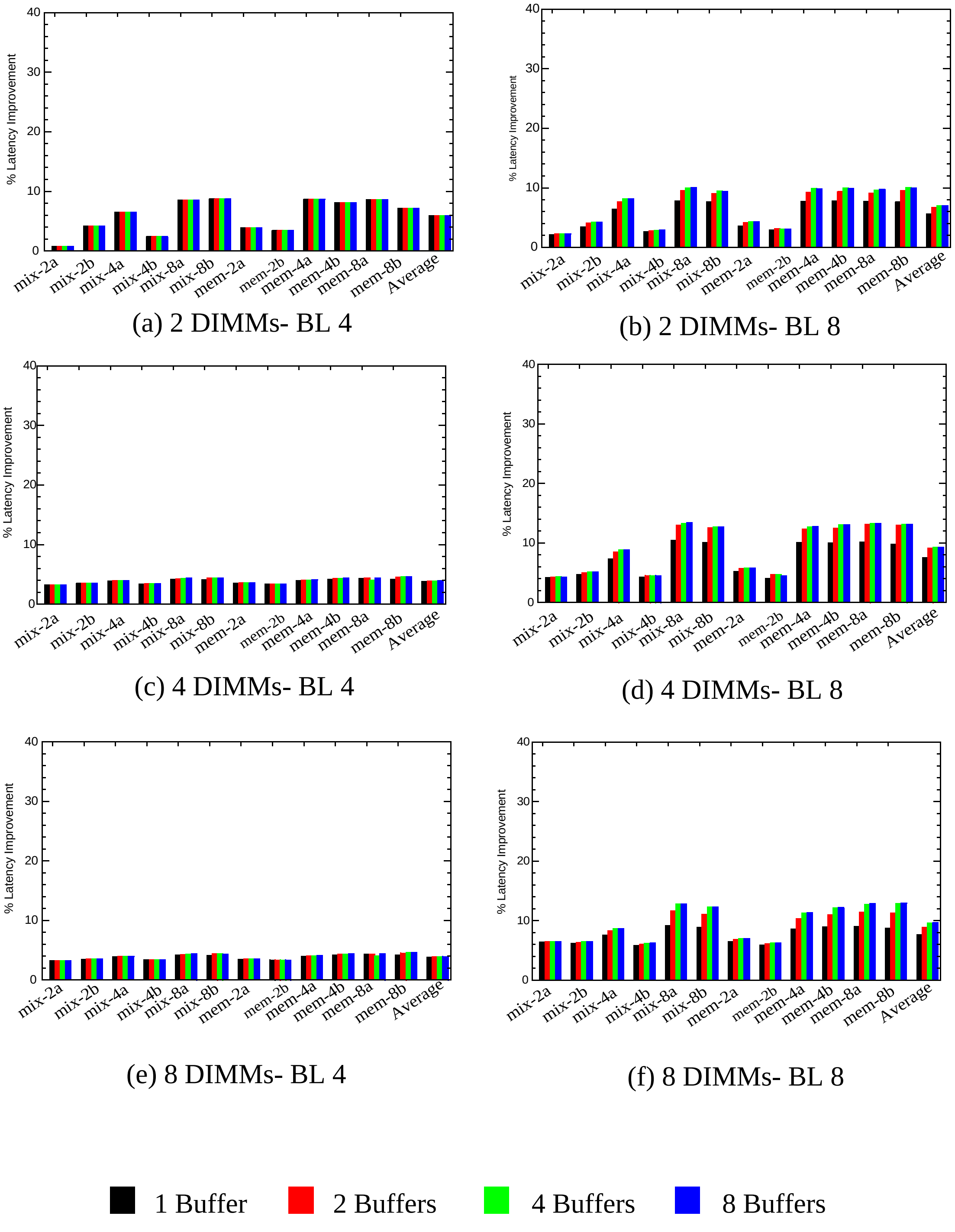
(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

(f) 8 DIMMs- BL 8

Figure 5.63. Latency Improvements by using different buffering schemes in a system permitting re-ordering of data returns. The graphs plot the percentage improvements for the average read latency on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with a bar for a different buffering scheme.



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

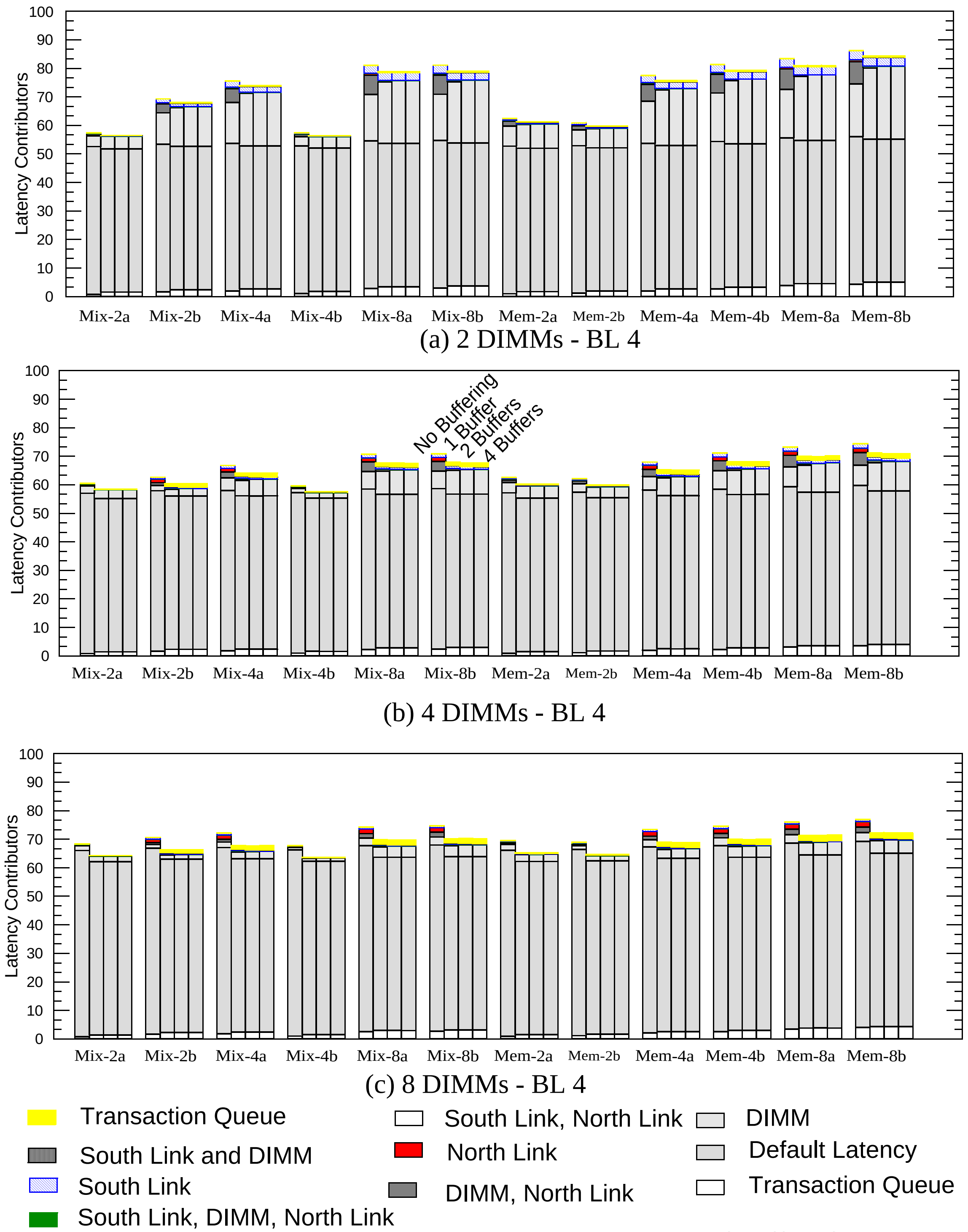
(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

(f) 8 DIMMs- BL 8

Figure 5.64. Impact of number of buffers on AMB on latency improvements by using Source Buffering and re-ordering. The graphs plot the percentage improvements for the average read latency on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with a bar for a different buffer count.

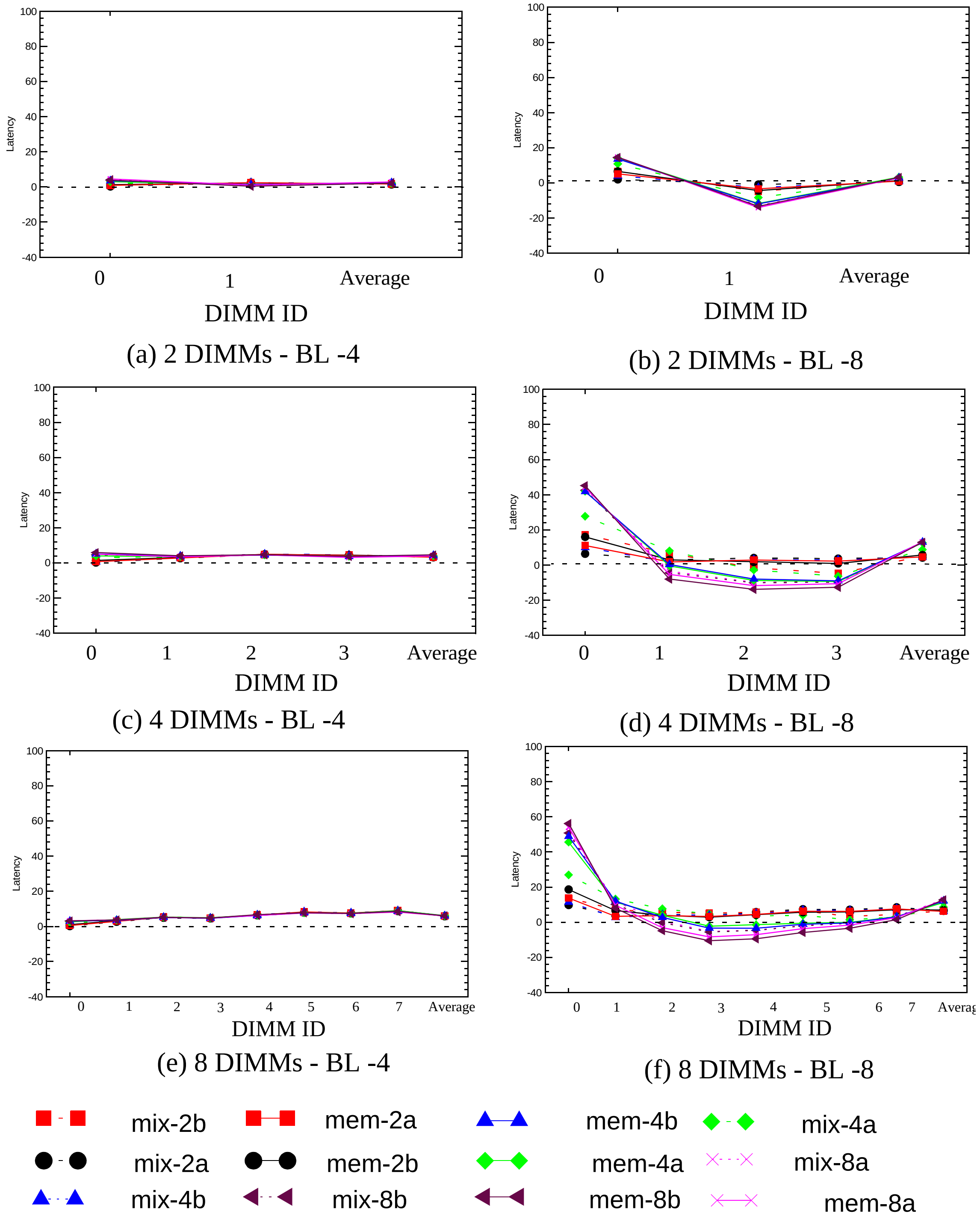


(a) 2 DIMMs - BL 4

(b) 4 DIMMs - BL 4

(c) 8 DIMMs - BL 4

Figure 5.65. Variation in Latency Contributors with number of buffers for system using re-ordering and buffering. The graphs are plotted for a single channel closed page FBD-DDR3 system which uses a burst length of 4, Source Buffering and supports re-ordering of data returns. The y-axis plots the latency in ns, split into its various contributing causes for the various workload combinations which are shown on the x-axis.



(a) 2 DIMMs - BL -4

(b) 2 DIMMs - BL -8

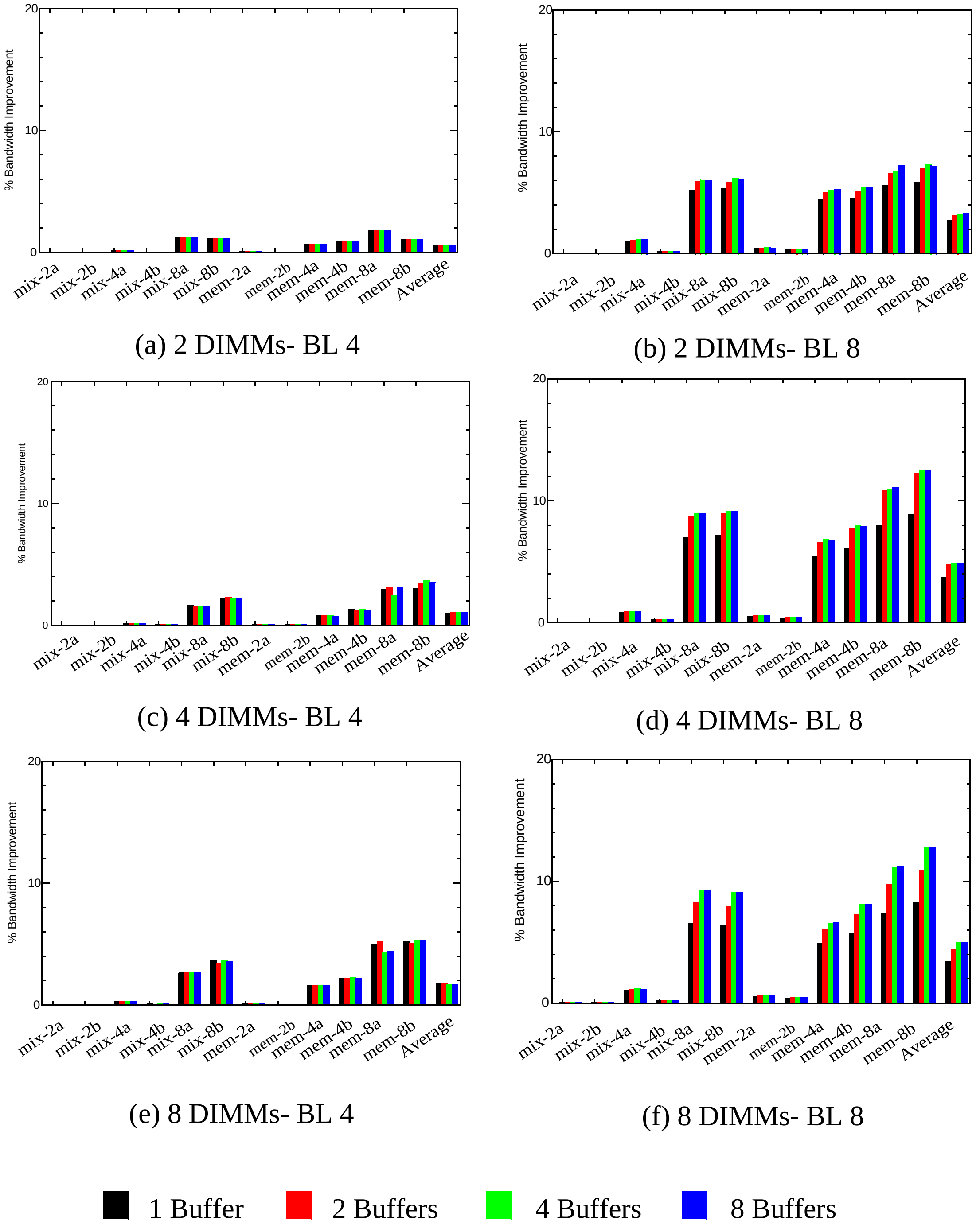
(c) 4 DIMMs - BL -4

(d) 4 DIMMs - BL -8

(e) 8 DIMMs - BL -4

(f) 8 DIMMs - BL -8

Figure 5.67. Improvements in Average Latency of a transaction to a given DIMM using Source Buffering in a system that supports re-ordering of read data returns. The graphs in the figure shows the variation in read latency for a transaction addressed to a particular DIMM in the system. On the y-axis is plotted the percentage improvement in latency for each DIMM's transactions. The various DIMMs are plotted on the x-axis, with the numbering starting with the DIMM closest to the memory controller. Note the last point is the overall average read latency improvement



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

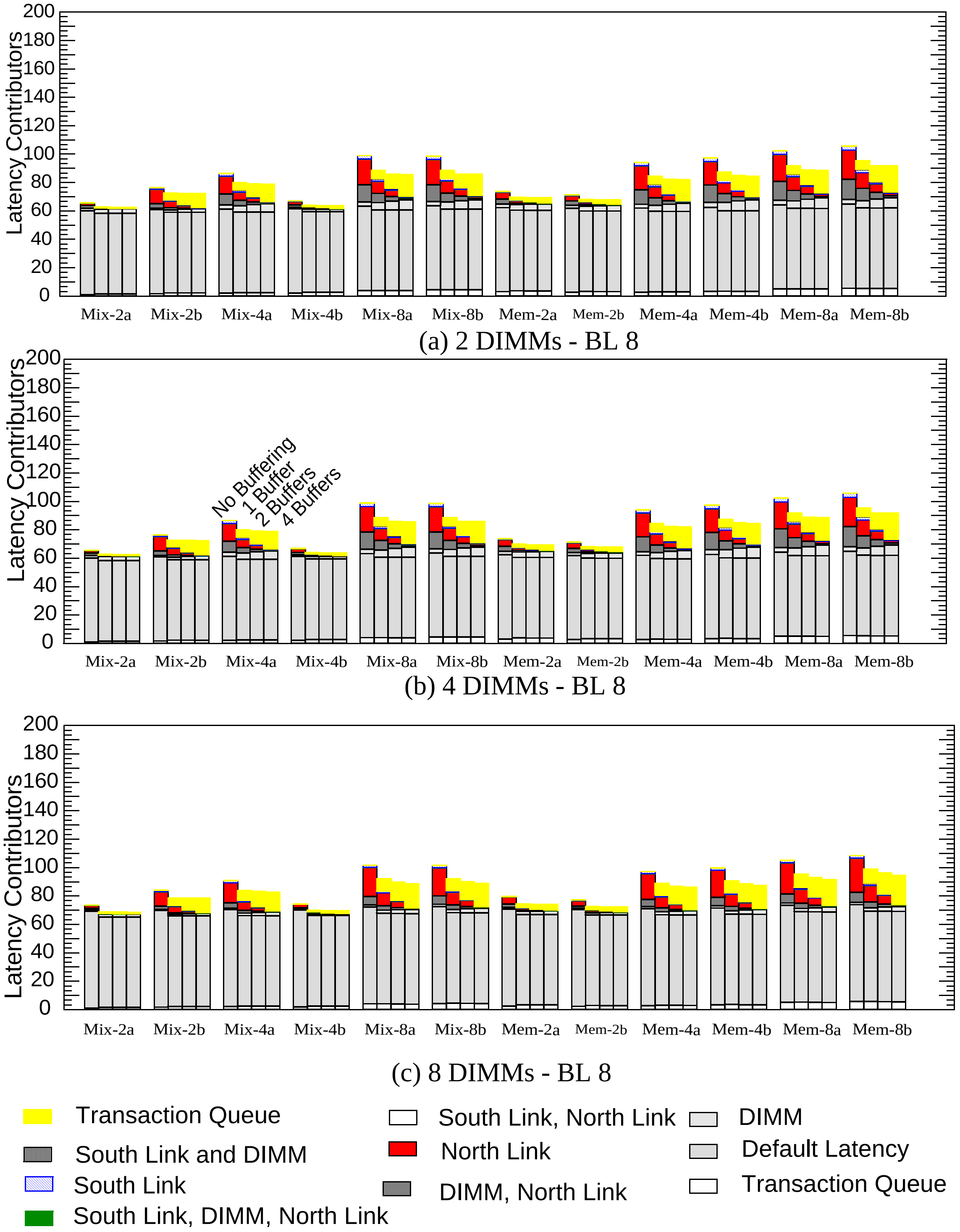
(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

(f) 8 DIMMs- BL 8

Figure 5.68. Bandwidth improvements by using different number of buffers in a system permitting re-ordering of data returns and Source Buffering. The graphs plot the percentage improvements in bandwidth on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with a bar for a different buffer count.

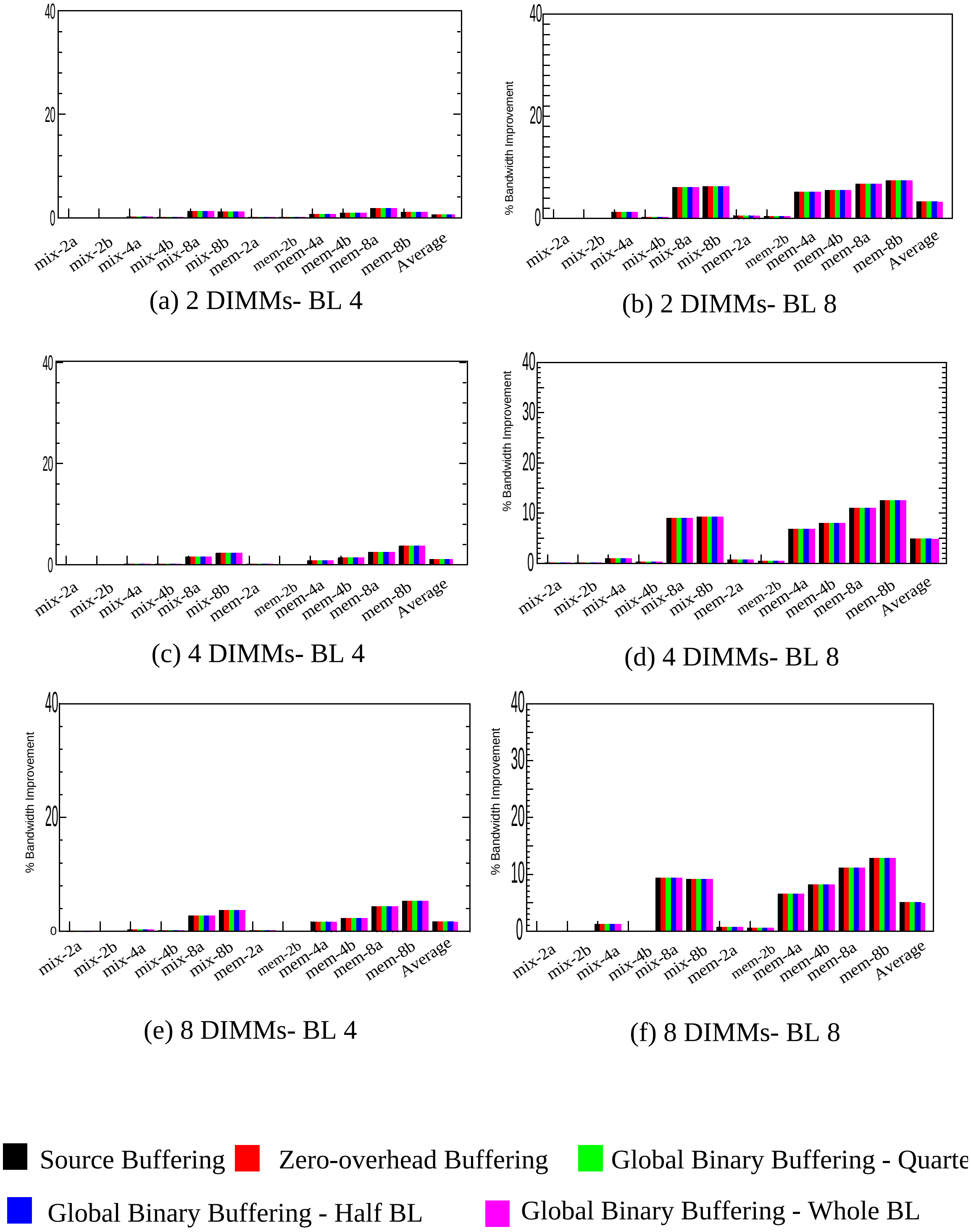


(a) 2 DIMMs - BL 8

(b) 4 DIMMs - BL 8

(c) 8 DIMMs - BL 8

Figure 5.66. Variation in Latency Contributors with number of buffers for system using re-ordering and buffering. The graphs are plotted for a single channel closed page FBD-DDR3 system which uses a burst length of 8, Source Buffering and supports re-ordering of data returns. The y-axis plots the latency in ns, split into its various contributing causes for the various workload combinations which are shown on the x-axis.



(a) 2 DIMMs- BL 4

(b) 2 DIMMs- BL 8

(c) 4 DIMMs- BL 4

(d) 4 DIMMs- BL 8

(e) 8 DIMMs- BL 4

(f) 8 DIMMs- BL 8

Figure 5.69. Impact of buffering type on bandwidth improvements for a system permitting re-ordering of data returns. The graphs plot the percentage improvements for the average read latency on the y-axis for the different applications at different channel depth. Each application's data is grouped on the x-axis with a bar for a different buffering schemes.

Since re-ordering is able to considerably improve the northbound link utilization, we see that a system which uses re-ordering is insensitive to the buffering policy employed. Re-ordering improves north link utilization by being able to insert the returning of read data in idle gaps on the channel which occur prior to the return of read data from a DIMM further away from the memory controller. By doing this, read transactions to earlier DIMM in the channel are able to utilize idle slots which are unavailable to transactions to later DIMMs as well as not add to the competition for slots after the data burst. Hence, re-ordering does not increase the latency of a transaction to any particular DIMM while significantly improving the read latency of transactions to the first DIMM. Hence, an ability to buffer at multiple places is no longer an advantage as in the case of a system using in-order returns and we see no benefit from varying the scheduling policy.

Another artifact of the above behavior is that buffering results in a latency degradation for transactions to further DIMMs in a system with burst length 8. This degradation is outweighed by the improvements in latency for transactions to mainly the first DIMM in the chain and an overall improvement in latency is seen.

Bandwidth improvements are more apparent in burst length 8 systems, due to their higher bus utilization. We find that using buffering not only moves data out of the DRAM faster but also packs the bus more efficiently contributing to an increase in the overall bandwidth of systems with higher bandwidth requirements i.e. applications running on a system with burst length 8 and applications with higher bandwidth requirements such as the memory intensive workloads

TABLE 5.2. Out of order Processor parameters

Parameter	**Value**
Width	8
Instruction Queue	64
Reorder Buffer Size	128
Load-Store Queue	32
Brach Predictor	Hybrid - 2K Local History/Return Address Stack 16/ Branch Target Buffer 2K/ GShare Predictor
D-L1	32KB / 3 cycles/ 16 MSHRs
I-L1	32KB/ 3 cycles/16 MSHRs
L2	0.5MB per core / 10 cycles/16 MSHRs
Operating Frequency	2.7 GHz

5.5. Full System Simulation Studies

5.5.1 Methodology

In this study, we used M5, a full system simulator developed at the University of Michigan[1] in stand-alone mode. The memory model of M5 has been modified to use DRAMsim. The details of the processor model are given in table 5.2. The DRAM parameters are similar to what we used earlier. Note that in this case the ratio of the CPU frequency to the memory frequency is lower than that used in the traces. We use a CPU frequency to DRAM channel data rate ratio of 2:1.

For this study we used combinations of SPEC 2000 [3] benchmarks. EIO traces of the benchmarks which were collected using SIM-EIO were used in this study. The execution phase of the benchmarks are identical to those used by Choi et al[2]. The workload combinations used in the study are given in table 5.3. The IPC values for the workloads are

TABLE 5.3. Workload Combinations Used

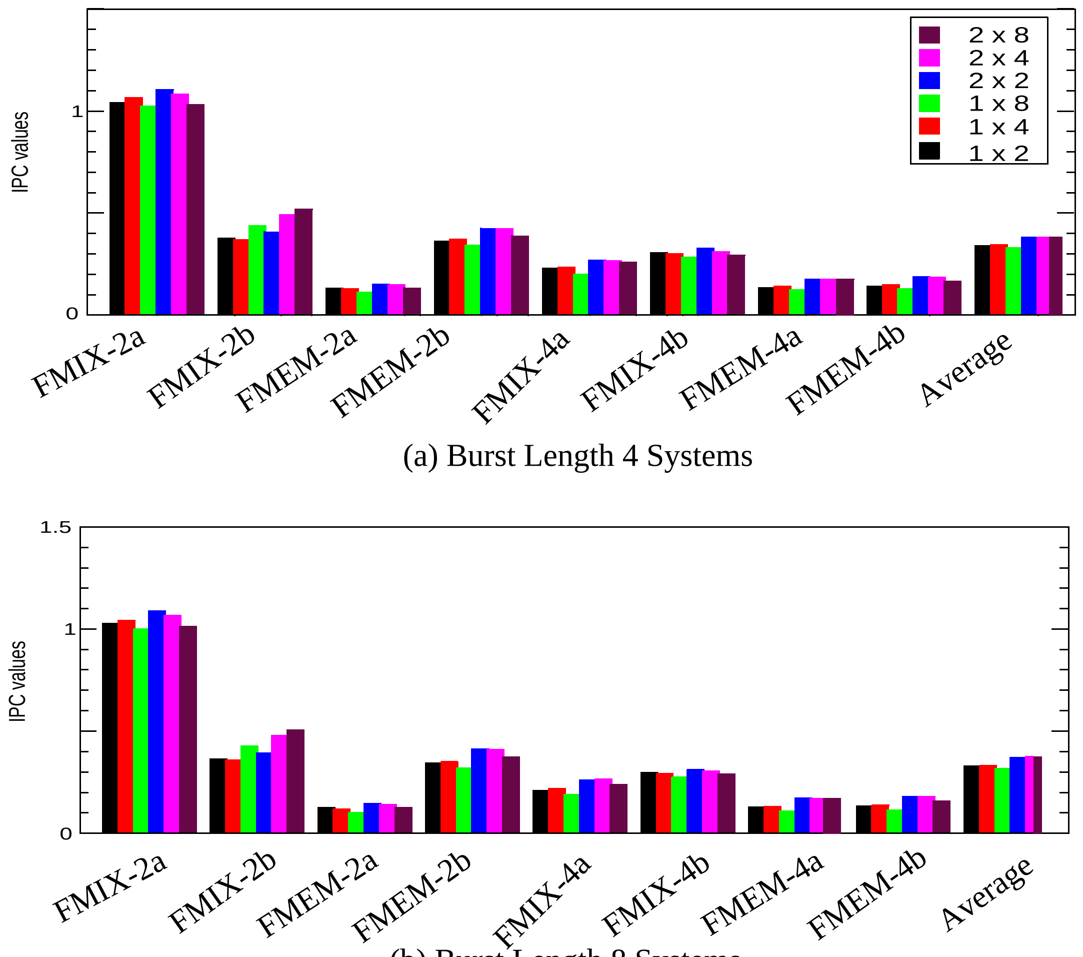
Name	Applications
FULL-MIX-2a	applu, vortex
FULL-MIX-2b	twolf,apsi
FULL-MEM-2a	art, mcf
FULL-MEM-2b	swim,twolf
FULL-MIX-2a	art, mcf, fma3d, gcc
FULL-MIX-2b	gzip, twolf, bzip2, mcf
FULL-MEM-2a	swim, twolf, art, mcf
FULL-MEM-2b	art, mcf, vpr, swim

given in figure 5.70 and the initial bandwidth used in figure 5.71. The bandwidth values were highest for the memory intensive workloads, with a bandwidth utilization of around 50-60% in the single channel case and 20-30% in a dual channel case. For nearly all the workloads, the observed bandwidth and IPC values decreased with increases in channel length.

5.5.2 Results

Figure 5.72 show the speed-up achieved by permitting return data to be re-ordered on the northbound channel. Figure 5.73 shows the improvements in IPC by using Source Buffering in a FBD-DDR3 closed page system. Figure 5.74 shows the improvements in IPC achieved by using both re-ordering of data returns and buffering.

For all techniques studied we found the most improvement in longer channel systems with higher bandwidth utilzations. A 2 DIMM channel saw nearly no speed-up, while a 8 DIMM deep channel saw a speed-up of 10-18% for the different policies. All the improvements were seen at higher bandwidth utilizations and hence were absent in systems with 2



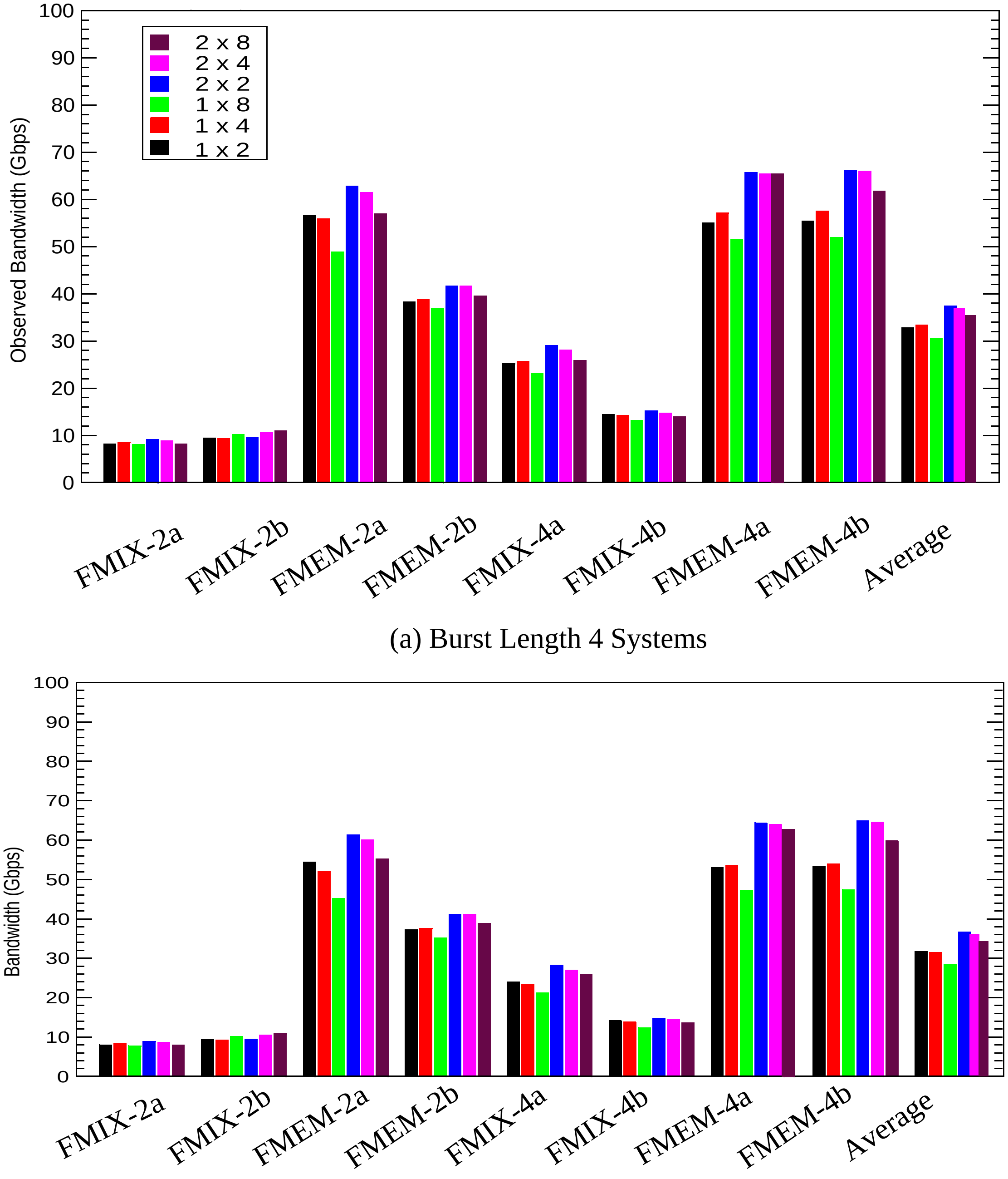
(a) Burst Length 4 Systems

(b) Burst Length 8 Systems

Figure 5.70. Default IPC values for Workloads. The default IPC values are plotted on the x-axis for the different workloads which are represented as groups on the y-axis.Each group is representative of a different system configuration.

or more channels. For the same reasons, the improvements were mainly seen for the memory intensive workloads and for workloads with 4 threads.

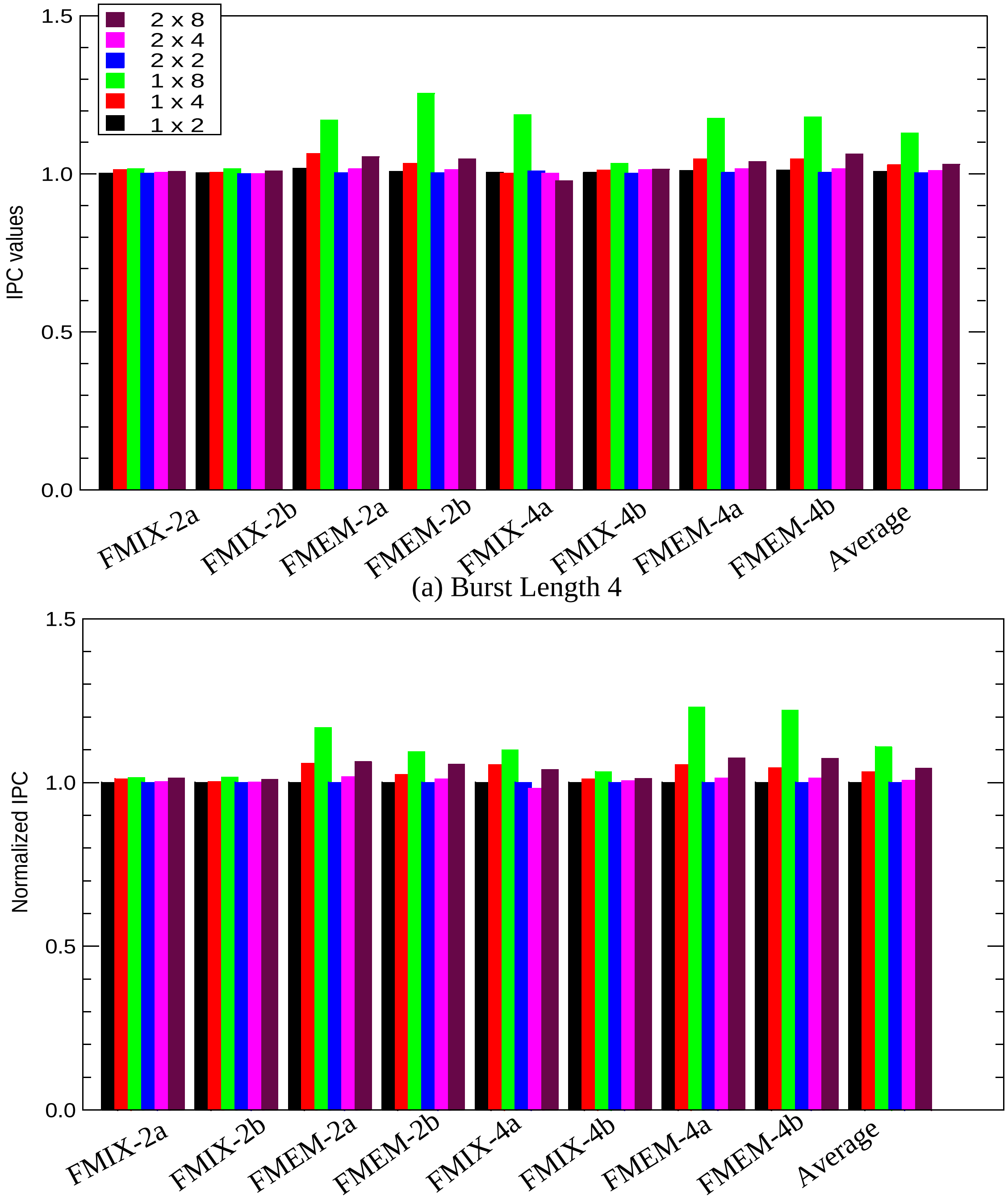
By using re-ordering of data returns alone, an average performance improvement of 10-12% for an 8 DIMM deep channel and around 5% for a 4 DIMM deep channel were seen. A maximum performance improvement of nearly 25% was seen for the FMEM-4a and FMEM-4b workloads running in a system using burst length 8. We found that in most cases the burst length 4 and burst length 8 systems had performance improvements which



(a) Burst Length 4 Systems

(b) Burst Length 8 Systems

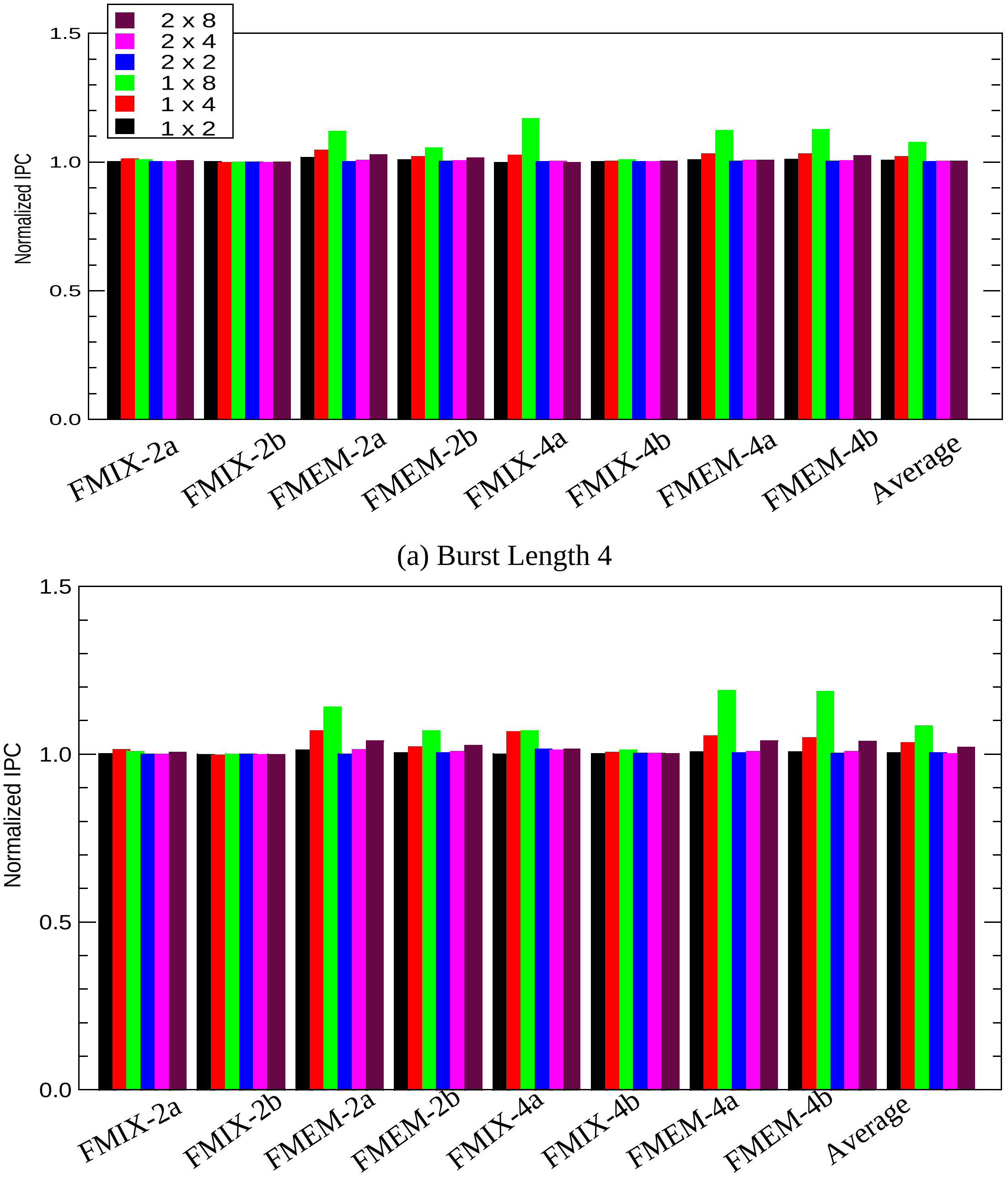
Figure 5.71. Default Sustained Bandwidth for Workloads. The observed bandwidth values are plotted on the y-axis for the different workloads which are represented ba a group of bars on the x-axis. Each group is representative of a different system configuration. The first three bars are for a single channel configuration, while the next three for a two channel configuration. Going from left to right, within a particular channel group, increases the number of DIMMs/channel.



(a) Burst Length 4

(b) Burst Length 8

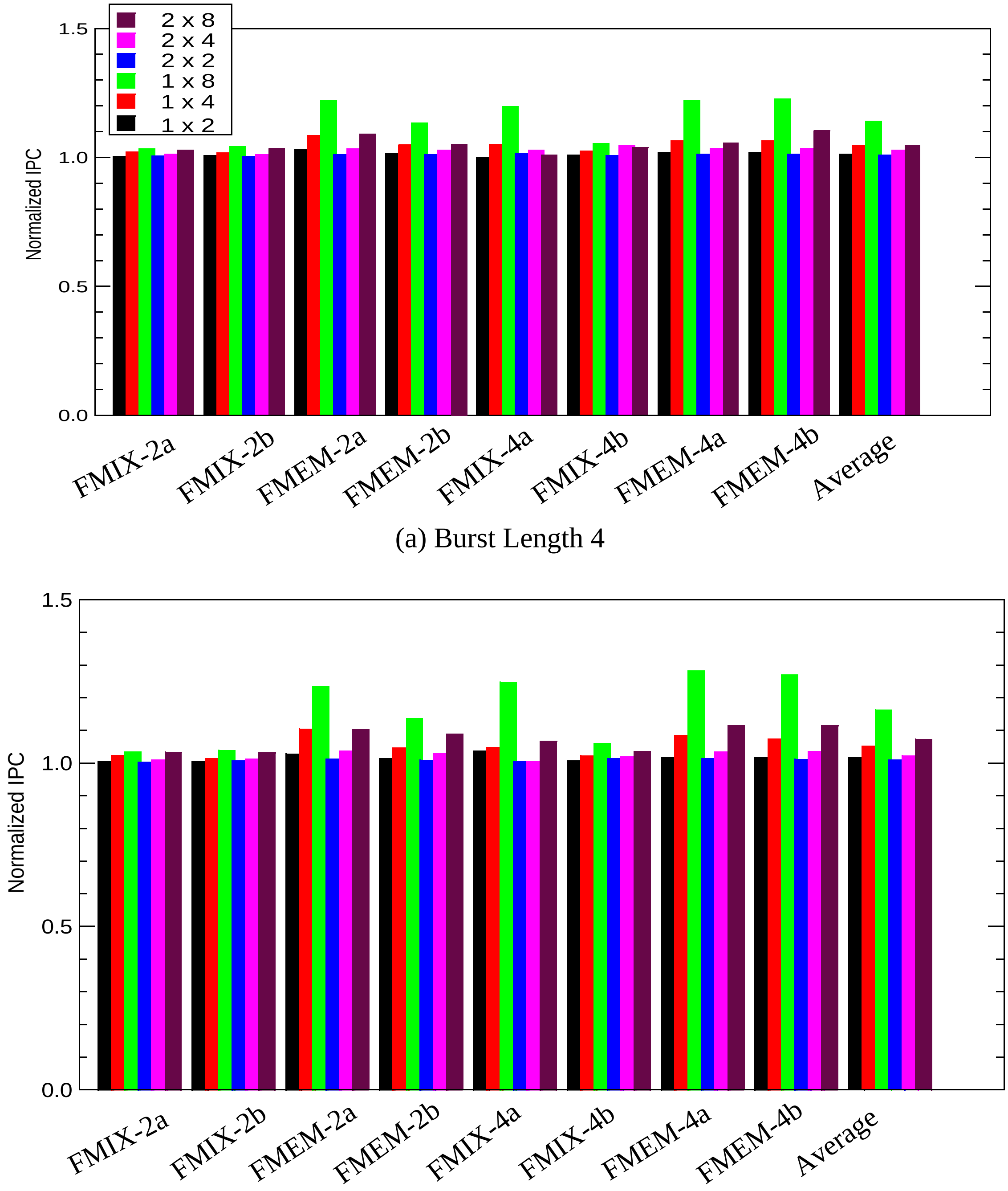
Figure 5.72. Improvements in IPC by using Re-ordering of Data Returns. The figure shows the improvements in IPC achieved by permitting re-ordering of data return and buffering for the various workloads studied, shown on the x-axis. Each set of bars is data for a different system topology for the workload. The topologies are specified as *#channels x # DIMMs*. The IPC values are normalized against the IPC values observed for the workload running on an identical system configuration with no re-ordering or buffering support.



(a) Burst Length 4

(b) Burst Length 8

Figure 5.73. Improvements in IPC by using Source Buffering. The figure shows the improvements in IPC achieved by permitting re-ordering of data return and buffering for the various workloads studied, shown on the x-axis. Each set of bars is data for a different system topology for the workload. The topologies are specified as *#channels x # DIMMs*. The IPC values are normalized against the IPC values observed for the workload running on an identical system configuration with no re-ordering or buffering support.



(a) Burst Length 4

(b) Burst Length 8

Figure 5.74. Improvements in IPC by using Buffering + Re-ordering of Data Returns. The figure shows the improvements in IPC achieved by permitting re-ordering of data return and buffering for the various workloads studied, shown on the x-axis. Each set of bars is data for a different system topology for the workload. The topologies are specified as *#channels x # DIMMs*. The IPC values are normalized against the IPC values observed for the workload running on an identical system configuration with no re-ordering or buffering support.

were within 5% of each other. The FMIX4a workload executing on a configuration using a burst length 4 and a bust topology with 2 channels each having 8 DIMMs saw a slight deterioration in IPC when re-ordering was used. This was mainly due to a 10% increase in the L2 cache miss rate which occurred when re-ordering was used. In nearly all the other cases, re-ordering did not impact the L2 cache rate significantly, and hence the improvements in memory latency translated into improvements in IPC.

Buffering again was most beneficial in a 8 DIMM deep channel achieving a speed-up of 10% for an 8 DIMM deep channel and 4% for 4 DIMM deep channels. For nearly all the workloads and configurations which showed performance improvement, buffering yielded slightly lower gain in performance than re-ordering data returns did.

The use of buffering and re-ordering resulted inhigher speed-ups than the use of the individual policies in burst length 8 systems. Burst length 8 systems benefitted by an additional 8% IPC improvement in a single channel with 8 DIMMs and 2% in a single channel with 4 DIMMs, for an overall improvement in IPC of 18 and 6% respectively.

5.6. Summary

Like previously defined memory protocols, the FBDIMM specification assumes an omniscient memory controller which manages all resources in the system, including the DRAM, on-DIMM buses i.e. command and data bus, and on-board links i.e. the south and north links. The memory controller has to guarantee that there are no violations of DRAM timing parameters or any conflicts on any of the various system buses. The default implementation of the protocol assumes that read data returns in the order that it was scheduled. Consequently, the scheduler is unable to take advantage of idle time on the north links which occur prior to the use of the link by a previously scheduled read transaction.

The first optimization that we looked at was to improve latency and bus utilization by relaxing the need for data to return in the same order as the commands. We propose a technique to permit re-ordering of returning read data that can be implemented without modifying the existing FBDIMM memory protocol. Permitting reordering of read data improves the maximum sustainable bandwidth of the system by 1-2 GBps. Application latency improves in a 4-8 DIMM deep system by an average of 10-20%, while bandwidth utilization increased by 5-10%. Multi-program workload runs on a full system simulator demonstrate that this technique improves overall IPC by an average of an average of 4 to 15%.

The second technique that we explored was focussed on de coupling DIMM availability from north link availability. We proposed using buffers on the northbound channel pat that hold read data frames. Buffering enables the memory controller to read data out of the DRAM rows without having to wait for the north link to become available. We examined several buffering policies including Source Buffering, Global Variable Buffering and Global Binary Buffering. These policies are distinguished by where buffering is permitted, i.e. at the DIMM supplying the data or at any DIMM on the path from the DIMM to the memory controller, and the buffering duration specified, a pre-defined fixed duration or a dynamically determined value. Buffering is managed by the memory controller and all buffering durations are specified by the memory controller in the command frames.

We found that buffering improves read latency by an average of 5 to 25% for channel lengths of 2 to 8 respectively. Most of these benefits arose from increasing the ability of the memory controller to move data out of the closest DIMM earlier. Buffering improved the overall IPC by an average of 2 to 10%.

The use of buffering with re-ordering of data returns was especially beneficial in a longer channel with higher burst lengths. This combination yielded an additional 10% reduction in latency and a further speed-up gain of 8% in an 8 DIMM deep FBDIMM channel using a burst length of 8.

In this chapter, we demonstrated that the inefficiencies in a long channel FBDIMM system operating in the variable latency mode can be eliminated by permitting read data to return in a different order than the requests were made in and by allowing the AMB to buffer the read data frame till the northbound channel is available. These techniques do not require significant modification of the current system design and can be easily implemented to regain the lost performance in long channel systems.

Chapter 6: Conclusions

Higher data-rates and increased DRAM capacity are often defined as the Moore’s Law of the memory system industry. Both these have been scaled successfully over the past decade. These scaling trends have become increasingly difficult to sustain due to the electrical limitations associated with scaling high-speed wide parallel bus architectures. To continue to meet the demand for increased memory capacity at higher bus speeds, the DRAM industry has converged on a high-speed, serial, split bus architecture, the Fully-Buffered DIMM (FBDIMM) standard.

In this dissertation, we first studied how this new bus architecture compared with the conventional bus architecture in terms of memory sub-system performance. Our study showed that the relative performance of a FBDIMM system and a DDRx system was a strong function of the bandwidth utilization of the input streams. Overall, the FBDIMM system had a 27% average higher latency, mainly due to workloads with bandwidth utilizations of less than 50% total DDRx DRAM bandwidth. This latency degradation becomes a latency improvement of nearly 10% for FBD-DDR3 systems as the application bandwidth utilization increased past 75% due to the ability of the FBD memory controller to use the additional DRAM level parallelism, split bus architecture and ability to send multiple DRAM commands in the same clock cycle. However, the additional system bandwidth available in a FBDIMM system resulted in an average of approximately 10% improvement in overall bandwidth with most of the benefits coming again for workloads with higher bandwidth utilization.

More interestingly, we found that the scheduling policies and row buffer management policies used in DDRx systems continued to perform comparably in FBDIMM systems. In both cases, a scheduling policy that prioritizes read traffic over write traffic had the best latency characteristics for both open page and closed page systems. A scheduling policy that prioritized traffic to currently open banks in the system had the best bandwidth characteristics while a greedy approach did very well in closed page systems.

One difference that we found with regard to FBDIMM and DDRx system behavior was their response to the use of posted CAS, a DRAM protocol feature which simplifies memory controller design by allowing the memory controller to bundle a row activation and read/write command in back-to-back command cycles. Unlike DDRx systems, FBDIMM systems using posted CAS had worse latency and bandwidth characteristics than systems which did not use this. This difference arose from the organization and use of the FBDIMM command frame and the sharing of the FBDIMM southbound bus by commands and write data.

We also performed a detailed analysis of how high-speed serial, multi-hop memory protocols, such as the FBDIMM, scale with changes in system topology and respond to changes in memory controller policies. Detailed measurements of the contributors to the read latency of a transaction revealed that a significant factor contributor to overall latency was delays associated with the unavailability of memory system resources, such as southbound channel, northbound channel and DRAM. Scaling the memory system configuration, by adding more ranks or channels, resulted in the unavailability of each of these factors varying in a different fashion and interacting in different ways to impact the observed latency. In general, we observed that short channel FBDIMM systems are limited by DRAM availability, while long channel FBDIMM systems are bound by channel bandwidth. This problem is exacerbated in variable latency mode configurations where significant latency and bandwidth degradation occur due to inefficient usage of the northbound FBDIMM channel.

The use of serialization and a multi-hop topology in FBDIMM systems has led to increases in the default cost of a read transaction. The FBDIMM protocol allows the channel to be configured in one of two modes, a fixed latency mode where the round-trip latency for a transaction is identical for all DIMMs and is set to the round-trip latency of the last DIMM in the chain. This mode imposes a higher default latency cost, which is not desirable. The alternate mode, known as the variable latency mode, has been provided to target this latency cost. In this mode, the channel is configured such that the round-trip latency of a transaction is a function of the distance of the DIMM from the memory controller. By allowing this, the FBDIMM protocol hopes to lower overall read latency.

Although the variable latency mode is able to reduce the average read latency for many of the workloads studied, this was not always true for applications executing in longer FBDIMM channels where the latency reductions were most needed. This problem was due to two reasons:- first, the inefficient utilization of the north link and second, the fact that transactions to closer DIMMs have to often wait for read data from further DIMMs to complete using the north link although the DRAM is ready.

We studied two techniques to improve the latency of an FBDIMM channel using the following techniques

- Out of order return of read data or allowing read data to return out of order,
- Buffers at the AMB for north link data to permit transactions to nearer DIMMs being issued in advance

Like previously defined memory protocols, the FBDIMM specification assumes an omniscient memory controller that manages all system resources, including the DRAM, on-DIMM buses i.e. command and data bus, and on-board links i.e. the south and north links. The memory controller has to guarantee that there are no violations of DRAM timing parameters or any conflicts on any of the various system buses. The default implementation of the protocol assumes that read data returns in the order that it was scheduled. Consequently, the scheduler is unable to take advantage of idle time on the north links which occur prior to the use of the link by a previously scheduled read transaction.

The first optimization that we looked at relaxed the requirement for data to return in the same order as the commands. We proposed a technique for re-ordering the returning read data that can be implemented without modifying the existing FBDIMM memory protocol. Permitting reordering of read data improves the maximum sustainable bandwidth of the system by 1-2 GBps. Application latency improves in a 4-8 DIMM deep system by an average of 5-25% while bandwidth utilization increased by 5-10%. Multi-program workload runs on a full system simulator demonstrate that this technique improves overall IPC by an average of 5-15%, with the most benefits being seen in a system with longer channels.

The second technique that we explored was focussed on de coupling DIMM availability from north link availability. We proposed using buffers on the northbound channel pat that hold read data frames. Buffering enables the memory controller to read data out of the DRAM rows without having to wait for the north link to become available. We examined several buffering policies including Source Buffering, Global Variable Buffering and Global Binary Buffering. These policies are distinguished by where buffering is permitted, i.e. at the DIMM supplying the data or at any DIMM on the path from the DIMM to the memory controller, and the buffering duration specified, a pre-defined fixed duration or a dynamically determined value. Buffering is managed by the memory controller and all buffering durations are specified by the memory controller in the command frames.

We found that buffering results in a speed-up of 2 to 25% for memory system topologies with 4 and 8 DIMMs per channel. Again, most of these benefits arose from increasing the ability of the memory controller to move data out of the closest DIMM earlier. A memory controller that uses buffering and re-ordering of data returns further reduces memory latency by an additional 10%. The overall speed-up by using both optimizations is on the order of 2-25%, over the baseline system which only allows in order return of data and does not support buffering.

References

Introduction

[1] J. Haas and P. Vogt. Fully-buffered dimm technology moves enterprise platforms to the next level. In *Technology@Intel Magazine*, March 2005.

[2] J. Henning. Spec cpu2000: Measuring cpu performance in the new millenium. In *IEEE Computer*, July 2000.

[3] P. Vogt. Fully buffered dimm (fb-dimm) server memory architecture: Capacity,performance, reliability, and longevity. Intel Developer Forum, Session OSAS008., Feburary 2004.

Related Work

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, April 1965.

[2] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *Computer Architecture News*, 1994.

[3] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," in *ISCA '81: Proceedings of the 8th annual symposium on Computer Architecture*, (Los Alamitos, CA, USA), pp. 81–87, IEEE Computer Society Press, 1981.

[4] D. Callahan, K. Kennedy, and A. Porterfield, "Software prefetching," in *ASPLOS-IV: Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, (New York, NY, USA), pp. 40–52, ACM Press, 1991.

[5] T.-F. Chen and J.-L. Baer., "A performance study of software and hardware data prefetching schemes.," in *ISCA*, 1994.

[6] D. Burger, J. R. Goodman, and A. Kagi, "Memory bandwidth limitations of future microprocessors," in *ISCA*, pp. 78–89, 1996.

[7] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, "A performance comparison of contemporary dram architectures," in *ISCA '99: Proceedings of the 26th annual international symposium on Computer architecture*, (Washington, DC, USA), pp. 222–233, IEEE Computer Society, 1999.

[8] Z. Zhang, Z. Zhu, and X. Zhang, “A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality,” in *MICRO 33: Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, (New York, NY, USA), pp. 32–41, ACM Press, 2000.

[9] S. Rixner, W. J. Dally, U. J. Kapasi, P. R. Mattson, and J. D. Owens, “Memory access scheduling,” in *ISCA*, pp. 128–138, 2000.

[10] S. Rixner, “Memory controller optimizations for web servers,” in *MICRO 37: Proceedings of the 37th annual International Symposium on Microarchitecture*, 2004.

[11] T. Takizawa and M. Hirasawa, “An efficient memory arbitration algorithm for a single chip mpeg2 av decoder,” in *ICCE International Conference onConsumer Electronics*, 2001.

[12] F. A. B. Chitra Natarajan, Bruce Christenson, “A study of performance impact of memory controller features in multi-processor server environment.,” in *Workshop on Memory performance issues*, 2004.

[13] D. T. Wang, *Modern DRAM Memory Systems: Performance Analysis and a High Performance, Power-Constrained DRAM-Scheduling Algorithm*. PhD thesis, University of Maryland College Park, 2005.

[14] B. D. Jun Shao, “A burst scheduling access reordering mechanism,” in *HPCA 07: Proceedings of the 13th International Symposium on High-Performance Computer Architecture*, Feburary 2007.

[15] W. fen Lin, S. K. Reinhardt, and D. Burger, “Reducing DRAM latencies with an integrated memory hierarchy design,” in *HPCA*, 2001.

[16] Z. Zhu and Z. Zhang, “A performance comparison of dram system optimizations for smt processors,” in *HPCA*, 2005.

[17] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, “High-performance drams in workstation environments,” in *IEEE Trans. Comput*, vol. 50, (Washington, DC, USA), pp. 1133–1153, IEEE Computer Society, 2001.

[18] Z. Zhu, Z. Zhang, and X. Zhang, “Fine-grain priority scheduling on multi-channel memory systems,” in *8th International Symposium on High Performance Computer Architecture, (HPCA-8)*, 2002.

[19] J. B. Carter, W. C. Hsieh, L. Stoller, M. R. Swanson, L. Zhang, E. Brunvand, A. Davis, C.-C. Kuo, R. Kuramkote, M. Parker, L. Schaelicke, T. T. Kuramkote, M. Parker, L. Schaelicke, and T. Tateyama, “Impulse: Building a smarter memory controller,” in *HPCA*, 1999.

[20] B. Mathew, S. McKee, J. Carter, and A.Davis, “Design of a parallel vector access unit for sdram memory systems,” in *Proceedings of 6th International Symposium on High-Performance Computer Architecture, 2000. HPCA-6.*, 2000.

[21] J. Shao and B. T. Davis, “The bit-reversal sdram address mapping,” in *SCOPES’05: Proceedings of the 9th International Workshop on Software and Compilers for Embedded Systems*, pp. 62–71, 2005.

[22] T. Mitra and T. cker Chiueh, “Dynamic 3d graphics workload characterization and the architectural implications,” in *International Symposium on Microarchitecture*, pp. 62–71, 1999.

[23] F. Harmsze, A. Timmer, and J. van Meerbergen, “Memory arbitration and cache management in stream-based systems,” in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition.*, 2000.

[24] K.-B. Lee, T.-C. Lin, and C.-W. Jen, “An efficient quality-aware memory controller for multimedia platform soc,” in *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 15, 2005.

[25] K.-B. Lee and C.-W. Jen, “Design and verification for configurable memory controller-memory interface socket soft ip,” in *Journal of Chin. Institute of Electrical Engineers*, vol. 8, p. 309 323, 2001.

[26] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith, “Fair queuing memory systems,” in *Proceedings of the 39th Annual International Symposium on Microarchitecture (MICRO-39)*, December 2006.

[27] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, “Dram energy management using software and hardware directed power mode control,” in *7th International Symposium on High Performance Computer Architecture*, 2001.

[28] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, “Scheduler based dram energy management,” in *DAC*, 2002.

[29] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis, “Power aware page allocation,” in *Architectural Support for Programming Languages and Operating Systems*, pp. 105–116, 2000.

[30] H. Huang, P. Pillai, and K. G. Shin, “Design and implementation of power-aware virtual memory,” in *USENIX Annual Technical Conference*, pp. 57–70, 2003.

[31] H. Huang, C. Lefurgy, K. Rajamani, T. Keller, E. V. Hensbergen, F. Rawson, and K. G. Shin, “Cooperative software-hardware power management for main memory,” in *4th International Workshop Power-Aware Computer Systems*, 2004.

[32] R. C. Schumann, “Design of the 21174 memory controller for digital personal workstations,” *Digital Tech. J.*, vol. 9, no. 2, pp. 57–70, 1997.

[33] K. M. Weiss and K. A. House, “Digital personal workstations: the design of high-performance, low-cost, alpha systems,” *Digital Technical Journal*, vol. 9, pp. 45–56, 1997.

[34] F. Briggs, M. Cekleov, K. Creta, M. Khare, S. Kulick, A. Kumar, L. P. Looi, C. Natarajan, S. Radhakrishnan, and L. Rankin, “Intel 870: A building block for cost-effective, scalable servers,” *IEEE Micro*, vol. 22, pp. 36–47, 2002.

[35] “Intel 975x express chipset.” http://www.intel.com/products/chipsets/975x/index.htm, 2005.

[36] S. Radhakrishnan and S. Chinthamani, “Intel 5000 series: Dual processor chipsets for servers and workstations.” Intel Developer Forum, 2006.

[37] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner, “Power5 system microarchitecture,” *IBM Journal of Research and Development*, vol. 49, 2005.

[38] C. Keltcher, “The amd hammer processor core.” Hot Chips 14, August 2002.

[39] C. Keltcher, K. McGrath, A. Ahmed, and P. Conway., “The amd opteron processor for multiprocessor servers.,” *IEEE Micro*, vol. 23, pp. 66–76, March-April 2003.

[40] A. Saulsbury, F. Pong, and A. Nowatzyk, “Missing the memory wall: the case for processor/memory integration,” in *ISCA ’96: Proceedings of the 23rd annual international symposium on Computer architecture*, (New York, NY, USA), pp. 90–101, ACM Press, 1996.

[41] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, “A case for intelligent ram,” *IEEE Micro*, vol. 17, no. 2, pp. 34–44, 1997.

[42] W. Bowman, N. Cardwell, C. Kozyrakis, C. Romer, and H. Wang, “Evaluation of existing architectures in iram systems,” in *Workshop on Mixing Logic and DRAM, 24th International Symposium on Computer Architecture*, 1997.

[43] D. Patterson, K. Asanovic, A. Brown, J. G. Richard Fromm, B. Gribstad, K. Keeton, C. Kozyrakis, D. Martin, S. Perissakis, R. Thomas, N. Treuhaft, and K. Yelick, “Intelligent ram (iram): the industrial setting, applications, and architectures,” in *Intelligent RAM (IRAM): the Industrial Setting, Applications, and Architectures*, October 1997.

[44] R. Fromm, S. Perissakis, N. Cardwell, C. Kozyrakis, B. McGaughy, D. Patterson, T. Anderson, and K. Yelick, “The energy efficiency of iram architectures,” in *The 24th Annual International Symposium on Computer Architecture*, June 1997.

[45] C. Kozyrakis, J. Gebis, D. Martin, S. Williams, I. Mavroidis, S. Pope, D. Jones, D. Patterson, and K. Yelick, “Vector iram: A media-oriented vector processor with embedded dram,” in *12th Hot Chips Conference*, August 2000.

[46] C. Kozyrakis, D. Judd, J. Gebis, S. Williams, D. Patterson, and K. Yelick, “Hardware/ compiler co-development for an embedded media processor,” *Proceedings of the IEEE*, vol. 89, pp. 1694 – 1709, 2001.

[47] C. Kozyrakis and D. Patterson, “Overcoming the limitations of conventional vector processors,” in *30th Annual International Symposium on Computer Architecture*, 2003.

[48] D. Judd, K. Yelick, C. Kozyrakis, D. Martin, and D. Patterson, “Exploiting on-chip memory bandwidth in the viram compiler,” in *Second Workshop on Intelligent Memory Systems*, 2000.

[49] Y. Kang, M. W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas, “Flexram: Toward an advanced intelligent memory system,” in *ICCD ’99: Proceedings of the 1999 IEEE International Conference on Computer Design*, (Washington, DC, USA), p. 192, IEEE Computer Society, 1999.

[50] S. Yoo, J. Renau, M. Huang, and J. Torrellas, “Flexram architecture design parameters,” Tech. Rep. 1584, Center for Supercomputing Research and Development (CSRD), 2000.

[51] J. Torrellas, L. Yang, and A.-T. Nguyen, “Toward a cost-effective dsm organization that exploits processor-memory integration,” in *Sixth International Symposium on High-Performance Computer Architecture (HPCA)*, January 2000.

[52] B. Fraguela, P. Feautrier, J. Renau, D. Padua, and J. Torrellas, “Programming the flexram parallel intelligent memory system,” in *International Symposium on Principles and Practice of Parallel Programming (PPoPP)*, June 2003.

[53] Y. Solihin, J. Lee, and J. Torrellas, “Adaptively mapping code in an intelligent memory architecture,” in *2nd Workshop on Intelligent Memory Systems*, November 2000.

[54] M. Oskin, F. T. Chong, and T. Sherwood, “Active pages: A model of computation for intelligent memory.,” in *International Symposium on Computer Architecture*, 1998.

[55] M. Oskin, J. Hensley, D. Keen, F. T. Chong, M. Farrens, and A. Chopra, “Exploiting ilp in page-based intelligent memory,” in *Proceedings of the International Symposium on Microarchitecture*, November 1999.

[56] M. Oskin, F. T. Chong, and T. Sherwood, “Activeos: Virtualizing intelligent memory,” in *International Conference on Computer Design (ICCD99)*, 1999.

[57] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCauley, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, “Die stacking (3d) microarchitecture,” in *International Symposium on Microarchitecture (MICRO)*, 2006.

[58] M.-C. Chiang and G. Sohi, “Evaluating design choices for shared bus multiprocessors in a througput-oriented environment,” in *IEEE Transactions on Computers*, vol. 41, March 1992.

[59] W. R. Bryg, K. K. Chan, and N. S. Fiduccia, “A high-performance, low-cost multiprocessor bus for workstations and midrange servers,” *hp journal*, vol. 47, 1996.

[60] L. ARM, “Amba specification,” 1999.

[61] IBM., “Coreconnect bus architecture,” 1999.

[62] S. Ltd., “Open core protocol specification,” 1999.

[63] K. Lahiri, A. Raghunathan, and G. Lakshminarayana., “Lotterybus: A new high-performance communication architecture for system-on-chip designs,” in *Proceedings of Design Automation Conference*, 2001.

[64] R. Lu and C.-K. Koh, “Samba-bus: A high performance bus architecture for system-on-chips,” in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,*, vol. 15, 2007.

[65] V. Cuppu and B. Jacob, “Organizational design trade-offs at the dram, memory bus, and memory controller level: Initial results.,” Tech. Rep. UMD-SCA-TR-1999-2, University of Maryland Systems and Computer Architecture Group, 1999.

Background

[1] A. Ailamaki. Database architectures for new hardware. tutorial International Conference on Very Large Data Bases, August 2004.

[2] D. Burger, J. R. Goodman, and A. Kagi. Memory bandwidth limitations of future microprocessors. In *ISCA*, pages 78–89, 1996.

[3] D. Callahan, K. Kennedy, and A. Porterfield. Software prefetching. In *ASPLOS-IV: Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, pages 40–52, New York, NY, USA, 1991. ACM Press.

[4] T.-F. Chen and J.-L. Baer. A performance study of software and hardware data prefetching schemes. In *ISCA*, 1994.

[5] R. Crisp. Direct rambus technology: The new main memory standard. *IEEE Micro*, 17:18–28, December 1997.

[6] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. A performance comparison of contemporary dram architectures. In *ISCA ’99: Proceedings of the 26th annual international symposium on Computer architecture*, pages 222–233, Washington, DC, USA, 1999. IEEE Computer Society.

[7] J. Gasbarro. The rambus memory system. In *IEEE International Workshop on Memory Technology, Design and Testing*, 1995.

[8] J. Haas and P. Vogt. Fully-buffered dimm technology moves enterprise platforms to the next level. In *Technology@Intel Magazine*, March 2005.

[9] H. Ikeda and H. Inukai. High-speed dram architecture development. In *IEEE JOURNAL OF SOLID-STATE CIRCUITS,*, volume 34, 1999.

[10] B. Jacob and D. Wang. Lectures for enee759h, 2005.

[11] D. Kroft. Lockup-free instruction fetch/prefetch cache organization. In *ISCA ’81: Proceedings of the 8th annual symposium on Computer Architecture*, pages 81–87, Los Alamitos, CA, USA, 1981. IEEE Computer Society Press.

[12] Micron. *240-Pin 512MB/1GB/2GB DDR2 SDRAM FBDIMM (DR FB x72) Features*, April 2005.

[13] Micron. *1Gb: x4, x8, x16 DDR SDRAM Features*, January 2006.

[14] P. Vogt. Fully buffered dimm (fb-dimm) server memory architecture: Capacity,performance, reliability, and longevity. Intel Developer Forum, Session OSAS008., Feburary 2004.

[15] J. von Neumann. First draft of a report on the edvac. *IEEE Ann. Hist. Comput.*, 15(4):27–75, 1993.

[16] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. Dramsim: A memory-system simulator.". *SIGARCH Computer Architecture News*, 33, 2006.

[17] W. A. Wulf and S. A. McKee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 1994.

[18] R. Yung, S. Rusu, and K. Shoemaker. Future trend of microprocessor design: Challenges and reality. In *28th European Solid-State Circuits Conference*, 2002.

Performance Evaluation

[1] S. Choi and D. Yeung. Learning-based smt processor resource distribution via hill-climbing. In *ISCA ’06: Proceedings of the 33rd annual international symposium on Computer Architecture*, pages 239–251, Washington, DC, USA, 2006. IEEE Computer Society.

[2] R. Desikan, D. Burger, S. Keckler, and T. Austin. Sim-alpha: a validated, execution-driven alpha 21264 simulator. Technical Report TR-01-23, The University of Texas at Austin, Department of Computer Sciences, 2001.

[3] J. Henning. Spec cpu2000: Measuring cpu performance in the new millenium. In *IEEE Computer*, July 2000.

[4] JEDEC. www.jedec.org.

[5] Micron. *240-Pin 512MB/1GB/2GB DDR2 SDRAM FBDIMM (DR FB x72) Features*, April 2005.

[6] Micron. *1Gb: x4, x8, x16 DDR SDRAM Features*, January 2006.

[7] S. Rixner, W. J. Dally, U. J. Kapasi, P. R. Mattson, and J. D. Owens. Memory access scheduling. In *ISCA*, pages 128–138, 2000.

[8] D. Tullsen and J. Brown. Handling long-latency loads in a simultaneous multithreaded processor. In *34th International Symposium on Microarchitecture*, 2001.

[9] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. Dramsim: A memory-system simulator.". *SIGARCH Computer Architecture News*, 33, 2006.

Optimizations for Variable Latency Mode

[1] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. "the m5 simulator: Modeling networked systems". *IEEE Micro*, 26(4):52–60, Jul/Aug 2006.

[2] S. Choi and D. Yeung. Learning-based smt processor resource distribution via hill-climbing. In *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*, pages 239–251, Washington, DC, USA, 2006. IEEE Computer Society.

[3] J. Henning. Spec cpu2000: Measuring cpu performance in the new millenium. In *IEEE Computer*, July 2000.

[4] S.-W. Moon, J. Rexford, and K. G. Shin. Scalable hardware priority queue architectures for high-speed packet switches. *IEEE Transactions on Computers*, 49(11):1215–1227, 2000.