Memory Controller Optimizations for Web Servers

Scott Rixner

Rice University
Houston, TX 77005
rixner@rice.edu

Abstract

This paper analyzes memory access scheduling and virtual channels as mechanisms to reduce the latency of main memory accesses by the CPU and peripherals in web servers. Despite the address filtering effects of the CPU's cache hierarchy, there is significant locality and bank parallelism in the DRAM access stream of a web server, which includes traffic from the operating system, application, and peripherals. However, a sequential memory controller leaves much of this locality and parallelism unexploited, as serialization and bank conflicts affect the realizable latency.

Aggressive scheduling within the memory controller to exploit the available parallelism and locality can reduce the average read latency of the SDRAM. However, bank conflicts and the limited ability of the SDRAM's internal row buffers to act as a cache hinder further latency reduction. Virtual channel SDRAM overcomes these limitations by providing a set of channel buffers that can hold segments from rows of any internal SDRAM bank. This paper presents memory controller policies that can make effective use of these channel buffers to further reduce the average read latency of the SDRAM.

1 Introduction

Network server performance has improved substantially in recent years, largely due to scalable event notification mechanisms, careful thread and process management, and zero-copy I/O techniques. These techniques use the processor and memory more efficiently, but main memory is still a primary bottleneck at higher performance levels. Latency to main memory continues to be a problem for both the CPU and peripherals because the speed of the rest of the system increases at a much faster pace than SDRAM latency decreases. Despite this continuing trend, very little emphasis has been placed on architectural techniques to reduce DRAM latency. Rather, significant effort has been spent developing techniques to minimize the number of DRAM accesses, to issue those accesses early, or to tolerate their latency. However, since DRAM technology improvements are unable to lower memory latency enough to keep up with increased processing speeds, an increasing burden is placed on these techniques to prevent DRAM latency from becoming even more of a performance bottleneck.

This work is supported in part by a donation from AMD.

This paper analyzes memory controller policies to reorder memory accesses to the SDRAM and to manage virtual channel SDRAM in order to reduce the latency of main memory accesses in modern web servers. SDRAMs are composed of multiple internal banks, each with a row buffer that caches the most recently accessed row of the bank. This allows concurrent access and lower access latency to rows in each row buffer. Virtual channel SDRAM pushes this concept further by providing a set of channel buffers within the SDRAM to hold segments of rows for faster access and greater concurrency. By providing more channels than banks in the SDRAM, more useful memory is available for fast access at any given time. However, the memory controller must successfully exploit the features of both conventional and virtual channel SDRAM in order to deliver high memory bandwidth and low memory latency.

In a modern web server with a 2 GHz processor and DDR266 SDRAM, memory references (including operating system, application, and peripheral memory traffic) arrive at the memory controller approximately once every 90–100ns. At this rate, a simple memory controller is able to satisfy both CPU and DMA accesses with an average read latency of 35–39ns for CPU loads and 24–27ns for DMA reads. In such a system, reordering accesses to the SDRAM does not reduce the average read latency despite the available locality and concurrency in the access stream. However, with virtual channel SDRAM, a memory controller that manages the channels efficiently and uses a novel design to restore updated segments can reduce the DRAM latency by 9–12% for CPU loads and by 12–14% for DMA reads.

As the disparity between CPU speed and DRAM latency increases, scheduling memory accesses and exploiting virtual channels will become more important in keeping memory latency low. When references arrive at the memory controller at a rate of approximately once every 40ns (still slightly higher than the average latency when the references arrive more slowly), then references begin to queue up in the memory controller. At this rate, memory access scheduling can reduce the average latency of CPU and DMA read accesses by up to 86%. This reduction in latency is accomplished almost entirely by overlapping operations within the internal SDRAM banks, as the row hit rate increases by less than 1% over sequential access. A memory controller that efficiently exploits virtual channel SDRAM, however, can achieve substantially higher segment hit rates, resulting in a reduction of the average CPU and DMA read latencies by 28–34% and 25–28%, respectively, below the lowest latencies that can be achieved with conventional SDRAM and memory access scheduling.

These results stress that as CPU speeds continue to increase at a faster pace than SDRAM latency decreases, incorporating latency reduction mechanisms into the memory controller is critical. A memory controller that exploits the structure of well designed SDRAM can dramatically reduce the achieved memory latency, which will lead to more efficient use of the CPU core and increased overall performance. Such mechanisms are complementary to existing latency reduction and tolerance techniques, and provide latency reductions that cannot be achieved by SDRAM technology improvements alone.

The following section discusses dual in-line memory modules (DIMMs) and the organization of modern SDRAM. Section 3 discusses the characteristics of the memory accesses that occur in a web server. Then, Section 4 explains the concepts of memory access scheduling and virtual channels. Section 5 discusses how the web server access traces were used to evaluate these mechanisms, and Section 6 analyzes their performance. Section 7 discusses related work, and Section 8 concludes the paper.

2 Modern DIMMs

A modern dual in-line memory module (DIMM) for high performance systems is composed of multiple double data rate (DDR) SDRAM chips. These SDRAMs are internally organized as multiple (typically four) independent memory banks. Each bank contains a two-dimensional array of memory cells that must be accessed an entire row at a time. An ACTIVATE command to the SDRAM moves the indicated row from the memory array to a dedicated row buffer for that bank. Row activation is a destructive operation, so the row must ultimately be written back to the bank. Once a row is active in the row buffer, any number of WRITE and READ commands can be issued in order to transfer columns of the active row into and out of the SDRAM. The width of each column is equal to the number of data pins, and columns are transferred on both the rising and falling edges of the DDR SDRAM clock. Furthermore, each READ or WRITE operation must transfer multiple columns of data, typically 2, 4, or 8. Finally, after the desired column operations are performed, a PRECHARGE command to the SDRAM restores the row in the indicated bank's row buffer back to the memory array and precharges the bank for the next row activation. A more detailed overview of several different modern SDRAM types and organizations as well as an analysis of the impact of main memory organization on system performance can be found in [4] and [5].

A DIMM is composed of several of these DDR SDRAMs that share address lines. The data lines of each SDRAM are concatenated with each other to form a wider data interface than any single SDRAM could support due to packaging constraints. Therefore, all of the SDRAMs perform exactly the same operations at the same time, effectively acting as one large, wide SDRAM. For example, the Kingston KVR266X72RC25 is a 1 GB PC2100 ECC DIMM which contains 18 DDR SDRAM chips [8]. The achievable access latency of such a DIMM is highly dependent on the amount of bank parallelism that is exploited and the number of column accesses per row access.

The latency of the SDRAM operations provides ample opportunity for exploiting parallelism among the banks. For example, a READ command could be issued to an active row of bank 0 on one cycle, and an ACTIVATE command could be issued to bank 1 on the next cycle. The read data transfer and row activation would then occur in parallel. Another READ command could be issued to bank 1 once the row is activated. If the timing is right, then the second read could begin to use the data pins as soon as the first read is complete. In this manner, parallelism can be exploited within the SDRAM to increase bandwidth and reduce latency.

3 Memory Behavior

The memory access pattern within a web server is determined by the operating system, web server application, and peripheral devices. Since modern web server applications actually have a very small memory footprint, especially those that utilize zero-copy I/O, the operating system and peripherals must be considered to get an accurate picture of memory system performance.

3.1 Web Server Traces

To analyze the behavior of the external memory in a web server, memory traces were collected from a functional full system simulation approximating a system composed of an AMD Athlon XP processor with 2 GB of physical memory and a 3Com 720024 Gigabit network interface card. The simulator accurately models the behavior of the system, but does not include any timing information. The simulated Athlon processor includes independent 2-way set associative L1 instruction and data caches, each with a capacity

Table 1. Web trace characteristics.

	CS	IBM	SPEC	WC
HTTP Throughput (Mbps)	934	634	358	907
Connections/second	1258	2030	1064	1790
Requests/second	3349	27447	2904	16549
Packets/second (Transmit)	87277	79445	35006	88932
Packets/second (Receive)	48222	48701	34616	52111
IPC	0.25	0.31	0.27	0.30

of 64 KB and a line size of 64 bytes. There is also a unified L2 cache that is 16-way set associative with a capacity of 256 KB and a line size of 64 bytes. All three caches use a write-back policy, and the L2 cache also performs next line prefetching on a miss. The traces were generated using Simics [11] running the FreeBSD 4.7 operating system and the event-driven Flash web server [16]. Flash uses `sendfile` for zero-copy I/O, `kqueue` for scalable event notification, and helper threads for disk I/O. The memory traces include all operating system, application, and peripheral traffic. For each trace, approximately one billion total memory operations were simulated after the system was warmed up, resulting in about 10 million SDRAM accesses.

Three actual web traces and a SPEC WEB99 run with 1000 connections (SPEC) were used to access the web server while the memory traces were collected. The three real traces are from the Rice Computer Science Department web site (CS), an IBM web site (IBM), and the 1998 Soccer World Cup web site (WC). The three real traces were replayed by *sclient*, a synthetic web trace replayer that uses an infinite demand model to stress the bandwidth capabilities of the server [1]. SPEC WEB99 uses a connection-oriented model to determine how many simultaneous connections a server can handle. These traces all have working set sizes that fit within 2 GB of memory, so they include very few disk accesses after the file cache is warmed up.

Table 1 shows the characteristics of the web traces when they access a real web server matching the configuration that was used to collect the memory traces. The server contains an Athlon XP 2600+, 2 GB of DDR SDRAM, and a 3Com 720024 Gigabit network interface card. The table shows the rate at which connections are established and HTTP requests are made, as well as the rate at which packets are sent and received by the server. The server achieves near the peak bandwidth of the Gigabit Ethernet link for the CS and WC traces. In contrast, the HTTP throughput is much lower for the IBM and SPEC traces. For the IBM trace, the processing required by the connection and request rates limits the achieved bandwidth. For the SPEC trace, the memory capacity and the processing to generate dynamic content limits the connection rate and the achieved bandwidth. The table also shows the achieved IPC of these benchmarks, measured using the Athlon's performance counters. These IPC values will be used to estimate the average time between SDRAM accesses in the traces.

Table 2. Breakdown of SDRAM accesses.

Trace	Instr. (1000s)	Access Type	Reads		Writes	
			1000s	MB	1000s	MB
CS	580,927	CPU	7,022	428.6	2,256	137.7
		DMA	2,266	132.6	159	6.4
IBM	602,714	CPU	5,151	314.4	1,732	105.7
		DMA	1,208	69.8	136	6.2
SPEC	569,254	CPU	6,478	395.4	1,780	108.6
		DMA	1,503	86.6	437	23.6
WC	533,967	CPU	5,136	313.5	1,790	109.3
		DMA	1,719	100.0	148	6.3

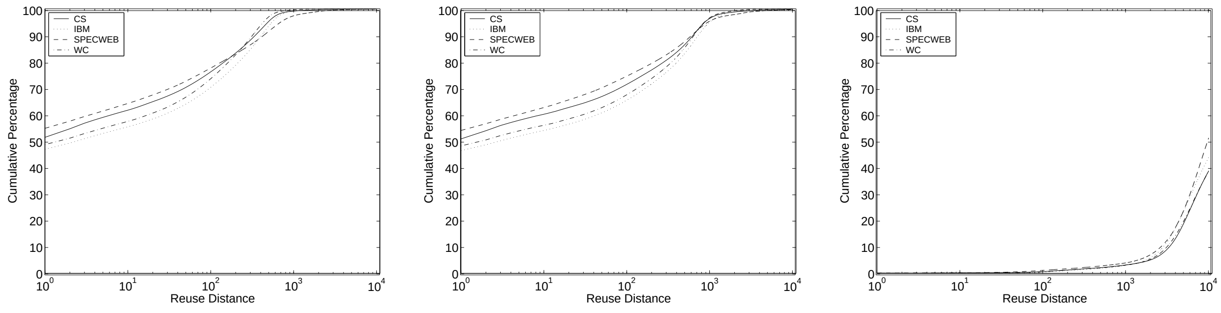
3.2 SDRAM Access Characteristics

Table 2 shows the characteristics of the SDRAM access stream for the four web server simulations. These traces include all memory activity from the entire system. As indicated by the table, approximately 80% of the SDRAM references originate at the CPU, with the remaining 20% coming from peripheral DMA transfers. Furthermore, read accesses dominate write accesses by a factor of 3.6. All CPU traffic is for 64B blocks (the L2 cache line size) whereas DMA transfers are for differing lengths. For three of the traces, 21–24% of the CPU read accesses are for instructions, and 35% are for instructions in the SPEC trace.

Figure 1 shows the cumulative distribution of the reuse distances for rows, segments, and 256 byte blocks of these SDRAM access traces. The reuse distance is the number of unique locations (row, segment, or block) accessed between accesses to a particular location. Since this data is collected using memory traces these accesses could occur in a slightly different order in a real system because of timing issues. However, the figure shows some very clear trends. Figure 1A shows the reuse distance for 32 KB blocks, which is equivalent to the row size in a 1 GB DIMM. For these rows, 51–59% of the accesses have reuse distances between 0 and 3, indicating that there is a good chance of getting a row hit in the SDRAM. Furthermore, about 6% more of the accesses have a reuse distance between 4 and 15, and fully 92–97% of the accesses have a reuse distance less than 512. So, there is significant locality in row accesses to SDRAM in actual web servers. In fact, less than 0.1% of the accesses in these traces were to rows that had not previously been accessed in the trace.

Figure 1B shows the reuse distance for segments, which is one quarter of a row (8 KB). Such segments are used in virtual channel SDRAM, which will be described in Section 4.2. The accesses with very short reuse distances remain about the same: 50–58% of the accesses have reuse distances between 0 and 3. However, only 83–88% of the accesses have a reuse distance less than 512, so there is slightly less segment locality than row locality.

Finally, Figure 1C shows the reuse distance for 256 byte blocks. Such blocks could be used to form a cache in the memory controller or SDRAM. As the figure shows, how-



(A) 32 KB Row Reuse **(B) 8 KB Segment Reuse** **(C) 256 B Block Reuse**

Figure 1. Cumulative distribution of SDRAM access reuse distances.

ever, most of the spatial reuse occurs at a much larger granularity. For instance, even a 2 MB fully-associative cache would have less than a 30% hit rate, since fewer than 30% of the accesses have reuse distances of 8092 or less. This shows that a small cache, either on the SDRAM chip or in the memory controller, would not be effective. Rather, a true L3 cache would be necessary to capture reuse using such small blocks. Furthermore, the graphs indicate that the larger block sizes (segments and rows) only reuse small portions of the data. So, constructing a cache outside of the SDRAM with large block sizes would waste critical SDRAM bandwidth.

4 Locality and Parallelism

Figure 1 suggests that a memory controller that sends accesses to the SDRAM in the order they arrive would achieve significant row hit rates. However, the figure also suggests that there is much additional locality available in the memory access stream. Furthermore, banks can be precharged and rows can be activated in the background, while the oldest reference is being serviced. Exploiting this parallelism can improve both the bandwidth and latency of the SDRAM, as about 40% of the accesses have row reuse distances greater than 15, limiting the benefits of exploiting locality alone. This section discusses two complementary techniques to increase locality and parallelism in the SDRAM: memory access scheduling and virtual channels.

4.1 Memory Access Scheduling

Several mechanisms exist to reorder memory references in order to increase locality and parallelism in SDRAM [3, 12, 17]. Briefly, a memory controller that performs memory access scheduling accepts memory accesses (from either the CPU or peripherals in a network server), buffers them, and sends them to the SDRAM in an order designed to improve the performance of the SDRAM. By taking advantage of the internal structure of the SDRAM, described in Section 2, significant improvements are possible. However, it is widely believed that these techniques are only applicable to the domains in which they were originally proposed—media processing and scientific computing—which exhibit stream oriented access patterns and have high latency tolerance. While the characteristics of the memory traffic are different for a network server than in those domains, Section 3 shows that there is significant locality in the memory traffic of a web server. This strongly suggests that these techniques can be effective.

Table 3 describes the scheduling algorithms considered in this paper. The *sequential*, *bank sequential*, and *first ready* schedulers are self-explanatory. However, the *row* and *column* schedulers also need a policy to decide whether or not to leave a row open within the row buffer after all pending accesses to the row have been completed. If the row is left open, future references to that row will be able to complete with lower latency. However, if future references target other rows in the bank, they will complete with lower latency if the row has been closed by a bank precharge operation. These schedulers will be prefixed with *open* or *closed* based on this completed row policy. Further details on these algorithms and the mechanisms for performing such reordering can be found in [12] and [17].

4.2 Virtual Channels

No matter how well a memory access scheduler reorders memory references, it cannot prevent a sequence of accesses from conflicting within a memory bank. When this occurs, performance will be limited if there are not enough accesses to other banks, as the latency of activating different rows within the bank will be exposed. Even with sufficient bank parallelism, the average access latency will still be increased over what it could have been if the accesses had hit in the active row. In order to keep pace with future systems, new memory architectures are needed to further reduce latency. As shown in Figure 1C, small caches on the SDRAM or within the memory controller will not help.

NEC developed virtual channel SDRAM in the late 1990s in order to alleviate the problem caused by bank conflicts [15]. Virtual channel SDRAM was marketed at the time as a solution for I/O intensive systems, since the I/O

Table 3. Memory Access Scheduling Policies.

Policy	Description
Sequential	Accesses are sent to the SDRAM in the order they arrive, which does not take advantage of any parallelism within the SDRAM.
Bank Sequential	The memory accesses are separated according to the SDRAM bank they target. For each bank, the accesses are still sent to the SDRAM sequentially. However, commands to multiple banks can be overlapped with each other, significantly increasing the parallelism within the SDRAM.
First Ready	An operation is sent to the SDRAM for the oldest pending memory access that is ready to do so. This increases parallelism and reduces latency by overlapping operations within the SDRAM.
Row	Row operations (either precharge or activate) that are needed for the pending accesses are sent to the SDRAM first. This reduces latency by initiating row operations as early as possible to increase bank parallelism. If no row operations are necessary, then column operations are performed.
Column	Column operations (either read or write) that will be needed for the pending accesses are sent to the SDRAM first. This reduces latency by transferring data into or out of the SDRAM as early as possible. If no column operations can be performed, either because the data pins are already in use or the appropriate rows are not activated, then row operations are performed.

and CPU access streams could potentially conflict with each other in the SDRAM banks. Virtual channels have also been proposed as one possible addition to the DDR2 standard [6]. Virtual channels provide additional storage on the SDRAM chip, which acts as a small row buffer cache that is managed by the memory controller. The memory controller can use these virtual channels to avoid bank conflicts, leading to lower achieved memory latencies.

Figure 2 shows the internal organization of a virtual channel SDRAM. The structure of the actual memory array remains unchanged, with multiple banks and a row buffer for each bank. However, column accesses may no longer be made from the row buffers. Instead, *segments* can be transferred between a row buffer and a channel. As shown in the figure, segments are one quarter of a row in the original NEC virtual channel SDRAM. A PREFETCH command transfers one segment out of an active row in a row buffer to one virtual channel. Once stored in a channel, column accesses may be made to the segment. The memory controller must return the segment to the appropriate row of the SDRAM, if it is modified, using the RESTORE command to transfer the virtual channel to one segment of a row in the row buffer. This potentially requires that row to be activated and then precharged after the RESTORE command. Note that if the bank is precharged after the PREFETCH command, then the row data is restored to the memory array, as in a conventional SDRAM. Therefore, segments that are read but not written never need to be transferred back to

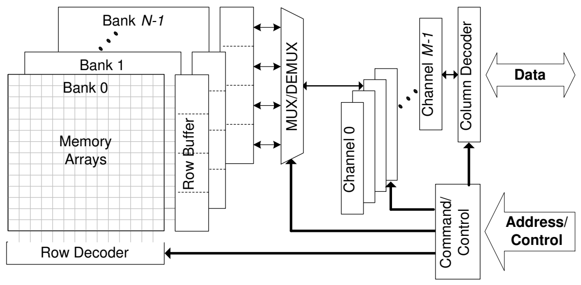


Figure 2. Virtual Channel SDRAM organization (N Banks and M Channels).

the row buffer, since the memory array already holds the correct data after the precharge operation. Furthermore, the bank may be precharged and other rows may be activated while a segment from that bank is in a channel. This allows each channel to service column accesses, regardless of the state of the bank from which it came. Furthermore, it allows opportunities for greater parallelism, as bank operations can occur in the background for all of the banks, while column operations are occurring to active segments.

NEC's original virtual channel SDRAM contained 64Mb organized into two banks of 8192 4Kb rows. This SDRAM had 16 channels that could each hold a single segment of 1Kb. NEC also included a dummy channel that allowed data to be written directly to a row buffer, bypassing the channels. Any SDRAM interface, such as double data rate or Rambus, could be augmented with virtual channels to improve access parallelism and locality.

4.3 Exploiting Virtual Channels

Unlike row buffers, which are specifically tied to a memory bank, virtual channels can be used to store any segment of the SDRAM. So, the memory controller must select a channel in which to store a segment before that segment can be accessed, and modified segments must be restored to the appropriate bank before the channel can be reused. The following proposed techniques effectively manage these segments in order to increase parallelism and reduce latency.

As with conventional SDRAM, virtual channel SDRAM will also benefit from memory access scheduling, since the underlying organization of the memory arrays remains similar. Scheduling accesses, however, now has the additional component of allocating channels. Most of the scheduling algorithms presented in Table 3 still apply to virtual channel SDRAM. However, the *row* policy would consider segment operations along with row operations first. Furthermore, the *bank sequential* algorithm is no longer useful, as column accesses to one segment of a bank can occur in parallel with row operations for another segment of the same bank, which was not possible with conventional SDRAM.

The memory access scheduling algorithm must be augmented with a channel selection algorithm, as all accesses utilize a virtual channel. Channel selection policies have not been explored in the past; rather, specific channels were dedicated to each memory consumer. However, fairly straightforward policies can be quite effective at reducing memory latency. Three effective policies are to use the channels in a round-robin fashion (*RR*), to use the least recently used channel (*LRU*), or to favor unmodified channels which do not need to be written back into the SDRAM banks (*Unmodified*). The *RR* policy does not consider locality at all, but rather simply exploits the large number of channels to improve parallelism and locality. The *LRU* policy utilizes the channels in a similar manner to a cache, but would require enough resources to track the use of all of the channels in the system, so is likely only practical for small numbers of channels. Finally, the *Unmodified* policy seeks to minimize the overhead of additional ACTIVATE, RESTORE, and PRECHARGE operations since unmodified channels may simply be overwritten with new segments.

The major drawback of virtual channel SDRAM is an increased worst case latency. If an access references a column which is not available in the row buffers of a conventional SDRAM, potentially a PRECHARGE, ACTIVATE, and column operation would be required. In a virtual channel SDRAM, however, potentially a PRECHARGE, ACTIVATE, PREFETCH, and column operation would be required. Since the SDRAMs are organizationally the same, the precharge, activate, and column latencies would be identical, so the additional latency of the prefetch operation is the only additional overhead. However, when a modified channel is selected, then that channel will need to be restored first, potentially requiring a PRECHARGE, ACTIVATE, RESTORE, and PRECHARGE operation. The increased latency to restore modified channels can significantly hamper performance. All of the channel selection policies suffer from this problem, to varying degrees, depending on the frequency of writes to the SDRAM and on whether unmodified channels are favored.

The decision of whether or not to restore a channel is similar to that of whether or not to keep a row open in conventional SDRAM. However, a segment is not lost when the channel is restored to the SDRAM. Therefore, modified channels can opportunistically be restored to the memory arrays when no other operations can be performed for the set of pending memory accesses. Dead memory controller cycles, in which no other commands could be given, are thereby effectively used to restore modified channels to the SDRAM banks. This can potentially completely hide the latency of restoring those channels without affecting the access latency of other references. This is similar to an eager writeback cache, in which dirty cache lines are written back to the DRAM before being evicted [9].

A memory controller enhanced with the novel policies described here can overcome the additional latency of virtual channels. The controller can keep multiple segments from the same bank active at the same time in order to capture additional locality in the memory access stream. Furthermore, row activations can occur while these segments are satisfying column accesses, leading to increased parallelism even when there are bank conflicts that would force serialization in a conventional SDRAM. This provides far greater opportunities to reduce latency than in a conventional SDRAM.

Table 4. DDR SDRAM parameters [14].

Parameter	Value	Units
Clock cycle (tCK)	7.5	ns
Precharge latency (tRP)	20	ns
Activate latency (tRCD)	20	ns
CAS (read) latency	2	cycles
Write latency (tDQSS)	1	cycle
Burst length	8	columns
Write to read delay (tWTR)	1	cycle
Active to precharge (tRAS)	40	ns
Activate to activate (tRRD)	15	ns
Write recovery time (tWR)	15	ns
Refresh latency (tRFC)	75	ns
Average refresh interval (tREFI)	7812.5	ns

5 Experimental Setup

A cycle accurate SDRAM simulator, *dsim*, written by the author was used to evaluate the performance of memory access scheduling and virtual channels for web servers. The simulator accurately models the behavior of the memory controller and the SDRAM, including all resource contention, latencies, and bandwidth limitations. The SDRAM model accurately simulates all timing parameters, including command latencies, all required delays between particular commands, and refresh intervals. The simulator is parameterized by the SDRAM timing parameters taken directly from the SDRAM data sheet. The memory controller within the simulator obeys all of these timing constraints when selecting commands to send to the SDRAM. The selected command is based upon the set of pending memory accesses and the particular policies that are being evaluated.

The experiments were run using a model of two 1 GB DIMMs, each consisting of 18 Micron MT46V128M4-75Z DDR SDRAMs [14]. The important timing parameters of this SDRAM are presented in Table 4. When part of a PC2100 DIMM, these DRAMs operate at 133 MHz and transfer data at 266 MHz. Two such 1GB DIMMs are the equivalent of a 72-bit wide DRAM (64 data bits and 8 ECC bits) with 8 internal banks, 4 from each DIMM. Physical addresses, which are 31 bits for the 2 GB address space, are mapped to the SDRAM as follows: the least significant 3 address bits are the offset into the 8 byte column, the next 12 bits are the column address, the next 3 bits are the bank address, and the 13 most significant bits are the row address.

Previous research has found that if the bank address is taken from bits that fall within the L2 set index, then write-backs will always cause bank conflicts [10, 19]. This is not an issue for such an Athlon-based system, as the set index and offset of the 256 KB, 16-way set associative cache with 64 byte lines only requires 14 bits. So, the bank address is already taken from the L2 tag. In systems where this is not true, the techniques described here should be augmented by using the XOR of bits from the row address with the bank address to generate a better bank mapping [10, 19].

To evaluate virtual channels, the Micron DDR SDRAM was modified to include virtual channels, using an original NEC virtual channel SDRAM data sheet as a guide [15]. A single set of unpipelined wires is available to transfer segments between the channels and the row buffers. The latency of such transfers was set at 2 cycles (1 cycle less than the activate latency of the SDRAM). A new segment operation, either a PREFETCH or RESTORE, can be initiated every 2 cycles. All other timing parameters remain the same. The virtual channel SDRAMs include 16 virtual channels. Each channel can hold a segment that is one fourth the size of a row (1024 4-bit columns per DRAM). The dummy write channel from the original NEC variants is not modeled.

The interaction between the operating system, web server application, and peripherals is quite complex and time varying. To understand the effects of any modifications to the memory system, the full system must be simulated in order to account for the activity from each of these three sources. No sufficiently detailed cycle accurate full system simulator exists that can boot an operating system and accurately model the behavior of peripherals. Regardless, such a simulator would be too slow to simulate any reasonable period of time. This would lead to significant inaccuracies, since the amount and type of memory traffic generated by the operating system, application, and peripherals varies greatly over time. To address this limitation, memory traces were collected from a functional full system simulator, as described in Section 3. These traces include all of the references made by the operating system, application, and peripherals.

Since the memory access traces were collected from a functional simulator, they contain no timing information. However, the rate at which memory references arrive at the memory controller can be estimated using the measured IPC values in Table 1, the instruction count of the simulations from Table 2, and the clock rate of the processor (2 GHz). Assuming that the measured average IPC corresponds to the average IPC of the simulated instructions, these simulations represent between 0.89 and 1.16 seconds of actual time. Given the number of references, this means that references arrive at the memory controller approximately once every 100 nanoseconds. Since the memory controller operates at the frequency of the SDRAM's data rate, 266 MHz, this corresponds to one access every 26 or 27 memory controller cycles.

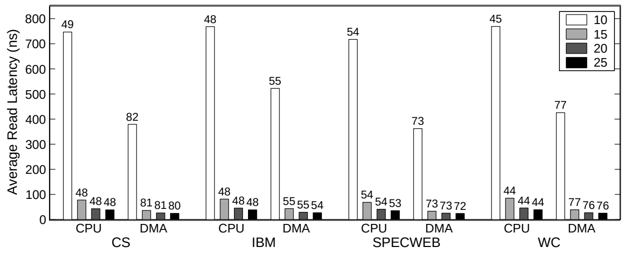


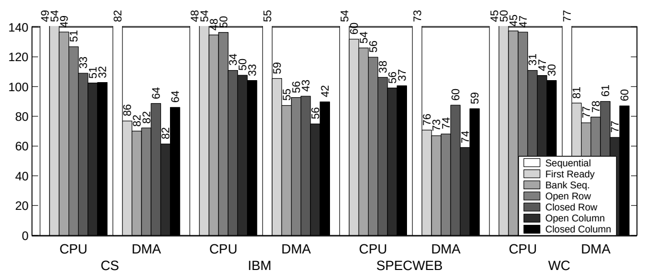
Figure 3. Average read latencies for sequential scheduling with inter-access times of 10–25 memory controller cycles. The achieved row hit rate is printed at the top of each bar.

To simulate the behavior of the memory system, the trace driven simulator feeds the accesses in the traces to the memory controller at a rate of once every 20 to 30 memory controller cycles. To approximate the variability of a real system, the inter-arrival time is randomly generated in that range. To show that these techniques become more important as the disparity between CPU speed and DRAM latency is increased, additional simulations show the average latencies when memory accesses arrive at the memory controller at the rate of one access every 15–25 cycles, 10–20 cycles, and 5–15 cycles. As CPU performance has increased at a faster rate than DRAM latency has decreased, these scenarios are likely predictors of future behavior. To smooth out any anomalies generated by randomly selecting inter-arrival times, all reported results are the average latencies of ten simulations using ten different random seeds.

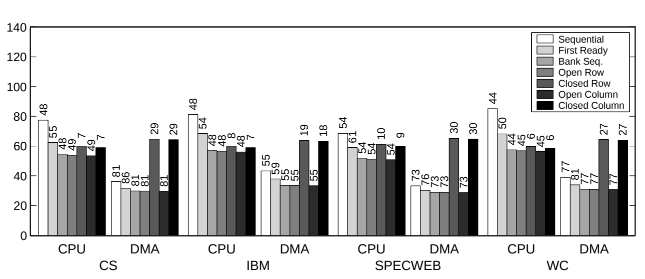
All latencies in the results are measured from the time an access arrives at the memory controller until the first data is transferred to or from the SDRAM for that access. The memory controller can handle up to 16 concurrent accesses, but if the memory controller cannot satisfy the accesses fast enough, the trace is stalled until resources become available.

6 Results

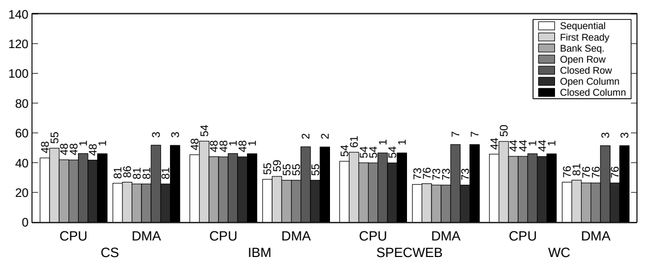
Figure 3 shows the baseline performance of a sequential memory controller on the web server memory traces when memory references arrive at the memory controller on average every 10, 15, 20, and 25 cycles. As the figure shows, the row hit rate for CPU accesses is 44–54% and the row hit rate for DMA accesses is 54–81%. This suggests that bank conflicts rarely prevent the memory controller from exploiting the short reuse distances found in Section 3.2. When accesses arrive at the memory controller every 25 cycles on average, the memory controller is able to achieve an average CPU read latency of 35–39ns and an average DMA read latency of 24–27ns. The DMA read latency is lower because



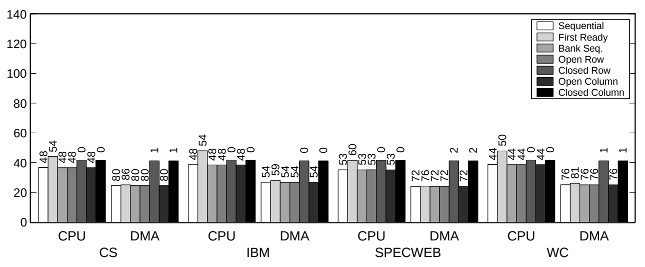
(A) Average inter-access time of 10 cycles



(B) Average inter-access time of 15 cycles



(C) Average inter-access time of 20 cycles



(D) Average inter-access time of 25 cycles

Figure 4. Average read latencies (ns) using memory access scheduling. In subfigure (A), the high latency of the sequential and first ready schedulers results in truncated bars. The achieved row hit rate is printed at the top of each bar.

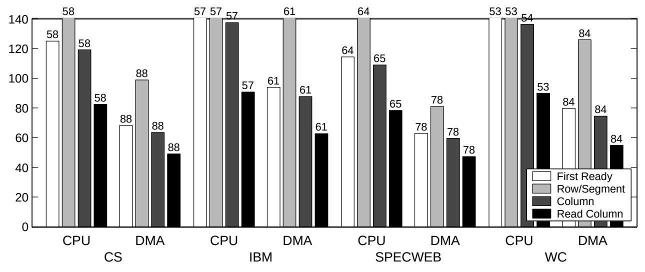
of the higher row hit rate achieved by sequential DMA accesses. As the average inter-access time decreases from 25 cycles to 15 cycles, the memory controller is still able to handle the access stream, but at an increased latency. The row hit rates remain largely the same, but queuing delays increase the average CPU read latency to as high as 81ns, and the average DMA read latency to as high as 43ns. When the average inter-access time decreases to 10 cycles, the memory controller and DRAM become overwhelmed, causing the average latencies to increase to as much as 766ns for CPU accesses and 520ns for DMA accesses. In a real system, the CPU and peripherals would slow down in response to the queuing delays in the memory controller. Therefore, the overall system would slow down significantly because of this memory bottleneck.

6.1 Memory Access Scheduling

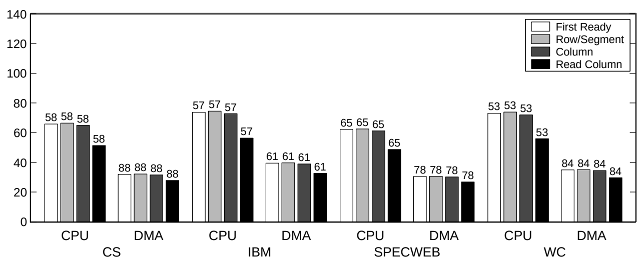
Figure 4 shows the effect of memory access scheduling on the read latency and row hit rate of the SDRAM. Six different scheduling policies are used in order to evaluate the effectiveness of the technique. For comparison, the baseline sequential scheduler is also shown in the figure. As the figure shows, the first ready scheduler achieves better row hit rates than the sequential scheduler. This leads to latency reductions with low inter-access times, but it actually increases the average latency for the longer inter-access times of 20 and 25 cycles. The latency increase at the slower access rates results from the fact that the first ready scheduler naively reduces the latency of some accesses by increasing the row hit rate, while significantly increasing the latency of others, causing a net increase in the average latency. While it does not increase the row hit rate, the very simple bank sequential scheduler reduces the average access latency by increasing bank parallelism within the DRAM. The latency is reduced more for the faster access rates, as there is more available parallelism. Quite significantly, unlike the sequential scheduler, both the first ready and bank sequential schedulers are able to handle the access rate for the case where the average inter-access time is 10 cycles, albeit with high average latencies.

The more aggressive scheduling policies are able to further reduce the average memory latency for the cases with low average inter-access times. As would be expected, the policies that leave rows open (*open row* and *open column*) are able to obtain up to about a 2% higher row hit rate than the baseline sequential controller, whereas the policies that aggressively close rows (*closed row* and *closed column*) achieve much lower hit rates, approaching 0% for the cases with high average inter-access times. In almost all cases, the policies that aggressively close rows yield significantly higher DMA read latencies. This is a direct result of the fact that DMA accesses have higher inter-access rates and are frequently sequential. So, rows must be held open after DMA accesses are completed in order to exploit the locality in the access stream. Similarly, the policies that keep rows open also benefit CPU latency, due to the reference locality found in Section 3.2.

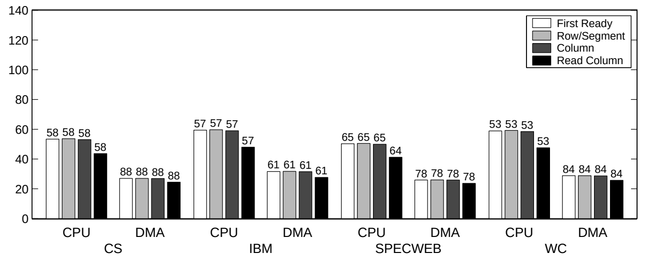
In almost all cases, the *open column* policy minimizes



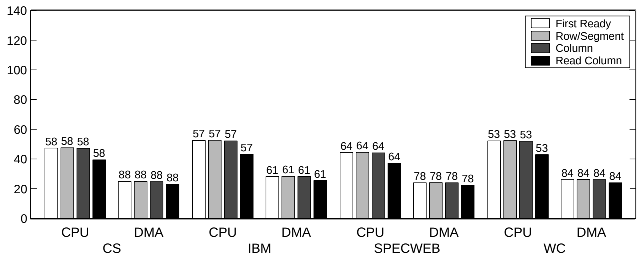
(A) Average inter-access time of 10 cycles



(B) Average inter-access time of 15 cycles



(C) Average inter-access time of 20 cycles



(D) Average inter-access time of 25 cycles

Figure 5. Average read latencies (ns) using virtual channels with an LRU channel selection policy. In subfigure (A), the high latency of several configurations results in truncated bars. The achieved segment hit rate is printed at the top of each bar.

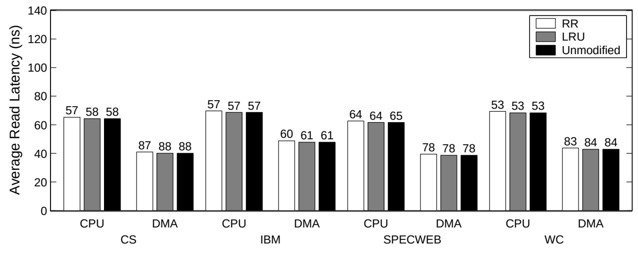
both the average CPU and DMA read latencies, so it is clearly the best policy across a wide range of traces and conditions. When the inter-access time averages 10 cycles, the *open column* scheduler reduces the average CPU read latency by 86% and the average DMA read latency by about 85% below the average read latencies achieved by the sequential scheduler. When the inter-access time increases to an average of 15 cycles, the *open column* scheduler reduces the average CPU and DMA read latencies by 26–31% and 14–23%, respectively, compared to the sequential scheduler. As the inter-access time increases further, the *open column* scheduler is only able to reduce the average read latencies by 3% at most, due to the decrease of available bank parallelism at any given time. So, while current systems will benefit very little from memory access scheduling, as CPU speeds increase relative to DRAM speeds, memory controllers will need to incorporate memory access scheduling to keep read latencies low.

The memory controller could give priority to read accesses over write accesses in an attempt to further reduce the average read latency. However, this is not that effective, as there is little or no effect when the average inter-access time is high, and at most a 10% improvement when the inter-access time is low.

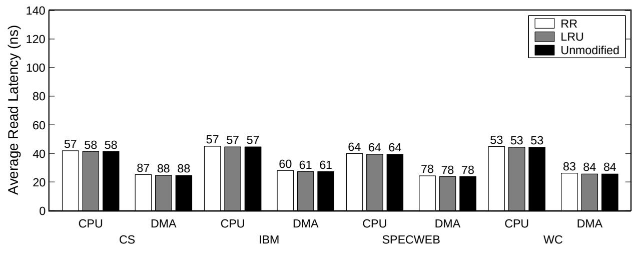
6.2 Virtual Channels

Aggressive memory access scheduling can potentially reduce the read latency of the SDRAM, but does not increase the row hit rate. Therefore, memory access scheduling reduces latency by overlapping operations to the SDRAM, not by exploiting the locality that was shown to exist in Section 3.2. This leaves an opportunity for virtual channel SDRAM to further reduce average latency. To evaluate the performance of virtual channel SDRAM, four memory access scheduling policies are used: *first ready*, *row/segment first*, *column first*, and column first giving priority to read operations (*read column*). Only the *LRU* channel selection policy is discussed here, as the *RR* scheduling policy performs slightly worse in all cases. Again, all memory access scheduling policies and channel selection policies are able to handle the access rate, except for the *row/segment first* policy at the highest access rate.

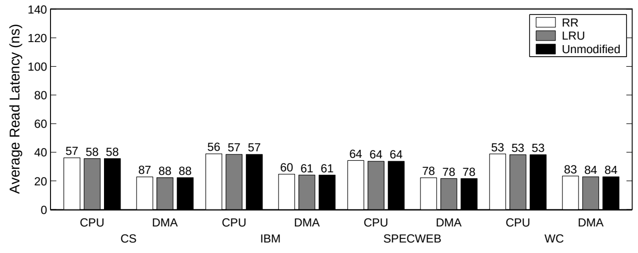
Figure 5 shows the achieved read latencies of virtual channel SDRAM. As expected from the reuse distances shown in Section 3, the segment hit rate is noticeably higher than the row hit rate of conventional SDRAM, even with the best scheduling policy. The *open column* policy achieved a 44–56% row hit rate for CPU accesses and a 54–82% row hit rate for DMA accesses, as shown in Figure 4. The best policy for virtual channel SDRAM, *read column* scheduling, achieves a 53–65% segment hit rate for CPU accesses and a 61–89% segment hit rate for DMA accesses, as shown in Figure 5. Therefore, virtual channels allow the memory controller to exploit additional locality beyond what is possible with conventional SDRAM. It would be difficult to capture this locality any other way. Bandwidth limitations prevent caching solutions that store segments outside of the SDRAM, and limited block reuse limits the utility of



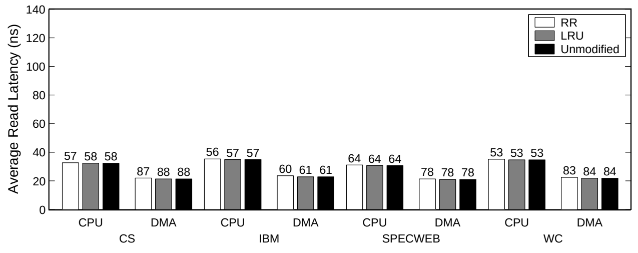
(A) Average inter-access time of 10 cycles



(B) Average inter-access time of 15 cycles



(C) Average inter-access time of 20 cycles



(D) Average inter-access time of 25 cycles

Figure 6. Virtual channel read latencies with read column scheduling and opportunistic channel restoration. The achieved segment hit rate is printed at the top of each bar.

caching data either within the SDRAM or the memory controller, as discussed in Section 3. However, despite the increased hit rate, the benefit of virtual channels over conventional SDRAM with the best scheduling policy is limited. When accesses arrive at the memory controller once every 10 cycles on average, *read column* scheduling reduces the average read latency by 5–16% for the CPU and by 2–11% for DMA accesses. When the access rate decreases, however, the average latency for CPU accesses increases over the best scheduling policy for conventional SDRAM by up to 13% in the worst case. In all cases, the average latency of DMA accesses is decreased, by at most 6%.

6.3 Channel Selection and Restoration

The schedulers evaluated in Section 6.2 are not able to achieve latencies as low as aggressive memory access scheduling with conventional SDRAM because of the increased latency required to restore modified channels before they can be reclaimed for reuse. In order to alleviate this problem, two techniques can be used, as described in Section 4.3. First, the channel selection algorithm can favor unmodified channels, so they do not have to be restored before reuse. Remember that the memory array within the SDRAM has already been restored, so channels can be overwritten if they do not contain new data. Second, modified channels can opportunistically be restored to the memory arrays when no other operations can be performed for the set of pending memory accesses.

Figure 6 shows the performance of virtual channel SDRAM with these optimizations to the memory controller. In all cases, a memory controller with these policies can handle the access rate. Furthermore, they also all significantly outperform the best scheduling policy with conventional SDRAM. Segment hit rate is not sacrificed by opportunistically restoring modified channels during free command cycles. This is because a RESTORE command to the SDRAM does not destroy the data in the channel. So, it remains available and may continue to be used until a new segment is moved to the channel via a PREFETCH command. However, by restoring segments in the background, a channel can be overwritten immediately when it is reclaimed. As the figure shows, this obviates the need for the channel selection policy to favor unmodified channels, as the *LRU* and *Unmodified* policies perform identically. The *Unmodified* policy offers no advantage because modified (dirty) channels get restored in the background before they become the least recently used channel. Furthermore, the simple *RR* policy performs almost as well as the *LRU* policy, so the complexity of LRU is unnecessary.

Recall from Figure 4 that with an average inter-access time of 25 cycles the *open column* policy achieves average CPU read latencies of 35–39ns and average DMA read latencies of 24–27ns. As Figure 6 shows, the *read column* scheduler with *LRU* channel selection for virtual channel SDRAM reduces these latencies by 9–12% for CPU reads to 30–35ns and by 12–14% for DMA reads to 21–23ns. As the inter-access time is decreased, such a memory controller is able to decrease the average read latency by even larger amounts compared to the best scheduling policy for conventional SDRAM. The average CPU read latency decreases by 12–15% when the inter-access time averages 20 cycles, by 17–20% when the inter-access time averages 15 cycles, and by 28–34% when the inter-access time averages 10 cycles. Similarly, the average DMA read latency decreases by 13–14% when the inter-access time averages 20 cycles, by 16–17% when the inter-access time averages 15 cycles, and by 25–28% when the inter-access time averages 10 cycles. In all cases, the average DMA read latency is lower than the average CPU read latency because of the greater access locality of DMA transfers.

With the appropriate access scheduling and channel selection policies, virtual channel SDRAM allows significantly lower average read latencies for web servers. Virtual channel SDRAM allows the memory controller to exploit increased amounts of locality in the access stream and parallelism within the SDRAM to reduce the average memory latency. As CPU and peripheral speeds increase, it becomes increasingly important to reduce SDRAM latency. Fortunately, the improvement becomes even more pronounced as the load on the memory system is increased.

7 Related Work

Several studies have proposed reordering memory accesses to increase effective memory bandwidth in stream-oriented systems [13, 17]. McKee, et al. show that for scientific and multimedia applications, a streaming memory controller can successfully reorder memory accesses to increase the row hit rate and therefore increase the achieved memory bandwidth [13]. Memory access scheduling also increases the achieved memory bandwidth of the Imagine media processor [17]. Both of these studies consider applications with high latency tolerance that are limited by memory bandwidth, so increasing the effective memory bandwidth improves performance in both cases. Neither study, however, considers the average latency of memory accesses and how to decrease them for applications which are not stream-oriented and have far less latency tolerance.

Several studies have also proposed to remap addresses and to utilize the cache hierarchy more efficiently in order to reduce the number of accesses to the DRAM, thereby improving its performance. Most notably, the Impulse memory controller attempts to improve the performance of the overall memory system by eliminating unnecessary DRAM accesses and increasing cache efficiency [2]. Such schemes do not necessarily improve the utilization of the SDRAM, but rather decrease the number of SDRAM accesses.

Impulse has also been augmented with a parallel vector access unit that allows commands to be sent to the memory controller. These commands expand to a stream of accesses and allow memory reordering similar to McKee's streaming memory controller [12]. Like the parallel vector access unit, the command vector memory system also sends commands to the memory controller that expand to a stream of accesses [3]. Additionally, the command vector memory system further reorders memory accesses to improve DRAM performance given the accesses generated by the commands. Again, however, the parallel vector access unit and command vector memory system focus on applications with high latency tolerance that are limited by memory bandwidth and do not consider average memory latency for latency limited applications.

There have been numerous proposals to add logic and storage to SDRAM devices. The two most relevant to this work are NEC's virtual channel DRAM and Wong and Baer's DRAM caching [15, 18]. NEC originally developed virtual channel DRAM to enhance traditional single data rate SDRAM and has also proposed the addition of virtual channels to DDR2 SDRAM [6, 15]. Davis has performed a limited evaluation of such virtual channel DDR2 SDRAM without exploring aggressive memory access scheduling and channel selection policies [7]. Wong and Baer propose the addition of a small cache on the DRAM itself to reduce latency. The reuse data in Figure 1 shows that such caches are far less effective than techniques such as virtual channels at exploiting locality. Furthermore, the memory controller has information about what references are pending, which makes it desirable for the memory controller to manage the memory on the SDRAM.

8 Conclusions

Operating system, application, and peripheral memory traffic within a web server combine to place considerable demands on the system's main memory. The need for low memory access latency and the internal structure of modern SDRAM motivate the use of aggressive techniques to exploit locality and concurrency within the SDRAM. By scheduling the accesses to the SDRAM to give priority to column accesses to open rows, the average read latency for both CPU and DMA reads can be reduced. In current systems, the reduction is modest. However, as CPU speeds increase and reduce the inter-access time to an average of 15 cycles, such a memory controller reduces CPU read latency by 26–34% and DMA read latency by 14–23%. Furthermore, when the inter-access time decreases to 10 cycles, the CPU and DMA read latencies can be reduced up to 86%. These decreases in latency are accomplished by increasing parallelism among the internal SDRAM banks, since the row hit rate increased by less than 1%.

Virtual channel SDRAM provides additional opportunities for latency reduction. First, virtual channels allow additional parallelism within the SDRAM. The effect of bank conflicts is minimized, as precharge and activate operations can occur to banks with open segments within the channels. Second, virtual channels allow the memory controller to capture additional locality within the SDRAM. By allowing multiple active segments from any bank, segments can remain active longer and more of the locality available within the access trace can be used to allow memory accesses to be satisfied with fast column accesses. A column first scheduling policy that favors read accesses along with a LRU policy to select virtual channels further reduces the average latency achieved below the best scheduling policy using conventional SDRAM. In order to offset the increased worst case latency of virtual channel SDRAM, however, channels must be opportunistically restored in the background. By doing so, in current systems, the average latency of CPU read accesses is reduced by 9–12%, and the average latency of DMA read accesses is reduced by 12–14% below the latency of conventional SDRAM. Also, as the inter-access times decrease, virtual channels are even more effective, yielding decreases of up to 28–34% in CPU read latency and 25–28% in DMA read latency. All of these latency reductions are accomplished both by increasing parallelism within the SDRAM and by increasing the segment hit rate beyond what was possible with conventional SDRAM.

With the appropriate access scheduling and channel selection policies, virtual channel SDRAM yields significantly lower average read latencies than conventional SDRAM. Such an external memory system can benefit a wide range of applications that have multiple competing outstanding accesses to DRAM. In the future, as CPU and peripheral speeds continue to rise, reducing DRAM latency will become increasingly important. To maximize system performance, techniques such as those described here must be employed, since technology constraints prevent memory speeds from keeping up with increases in processor performance. As the results in this paper have shown, the novel polices introduced for accessing and managing resources on the SDRAM continue to be effective as the demands on external memory increase. Furthermore, these mechanisms are complementary to existing latency reduction and tolerance techniques, and should increase the effectiveness of those techniques.

References

[1] G. Banga and P. Druschel. Measuring the capacity of a web server. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.

[2] 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, and T. Tateyama. Impulse: Building a smarter memory controller. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, 1999.

[3] J. Corbal, R. Espasa, and M. Valero. Command vector memory systems: High performance at low cost. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 1998.

[4] V. Cuppu and B. Jacob. Concurrency, latency, or system overhead: Which has the largest impact on uniprocessor DRAM-system performance? In *Proceedings of the International Symposium on Computer Architecture*, 2001.

[5] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. A performance comparison of contemporary DRAM architectures. In *Proceedings of the International Symposium on Computer Architecture*, 1999.

[6] B. Davis, T. Mudge, B. Jacob, and V. Cuppu. DDR2 and low latency variants. In *Proceedings of Solving the Memory Wall Workshop*, 2000.

[7] B. T. Davis. *Modern DRAM Architectures*. PhD thesis, The University of Michigan, 2001.

[8] Kingston. KVR266X72RC25/1024 memory module specification, June 2002.

[9] H.-H. S. Lee, G. S. Tyson, and M. K. Farrens. Eager write-back - a technique for improving bandwidth utilization. In *Proceedings of the International Symposium on Microarchitecture*, 2000.

[10] W. Lin, S. K. Reinhardt, and D. Burger. Reducing DRAM latencies with an integrated memory hierarchy design. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2001.

[11] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, and J. Hogberg. Simics: A full system simulation platform. *IEEE Computer*, February 2002.

[12] B. K. Mathew, S. A. McKee, J. B. Carter, and A. Davis. Design of a parallel vector access unit for SDRAM memory systems. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2000.

[13] S. A. McKee, W. A. Wulf, J. H. Aylor, R. H. Klenke, M. H. Salinas, S. I. Hong, and D. A. B. Weikle. Dynamic access ordering for streamed computations. *IEEE Transactions on Computers*, November 2000.

[14] Micron. 512Mb: x4, x8, x16 DDR SDRAM MT46V128M4 data sheet, July 2003.

[15] NEC. 64M-bit Virtual Channel SDRAM data sheet, October 1998.

[16] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *Proceedings of the USENIX 1999 Annual Technical Conference*, 1999.

[17] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *Proceedings of the International Symposium on Computer Architecture*, 2000.

[18] W. A. Wong and J.-L. Baer. DRAM caching. Technical Report UW-CSE-97-03-04, University of Washington, Department of Computer Science and Engineering, 1997.

[19] Z. Zhang, Z. Zhu, and X. Zhang. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *Proceedings of the International Symposium on Microarchitecture*, December 2000.