

Copyright © 1999 IEEE. Reprinted from the Proceedings of the 26th International Symposium on Computer Architecture (ISCA).

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the University of Maryland's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org.

By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

A Performance Comparison of Contemporary DRAM Architectures

Vinodh Cuppu, Bruce Jacob
Dept. of Electrical & Computer Engineering
University of Maryland, College Park
{ramvinod,blj}@eng.umd.edu

Brian Davis, Trevor Mudge
Dept. of Electrical Engineering & Computer Science
University of Michigan, Ann Arbor
{btdavis,tnm}@eecs.umich.edu

ABSTRACT

In response to the growing gap between memory access time and processor speed, DRAM manufacturers have created several new DRAM architectures. This paper presents a simulation-based performance study of a representative group, each evaluated in a small system organization. These small-system organizations correspond to workstation-class computers and use on the order of 10 DRAM chips. The study covers Fast Page Mode, Extended Data Out, Synchronous, Enhanced Synchronous, Synchronous Link, Rambus, and Direct Rambus designs. Our simulations reveal several things: (a) current advanced DRAM technologies are attacking the memory bandwidth problem but not the latency problem; (b) bus transmission speed will soon become a primary factor limiting memory-system performance; (c) the post-L2 address stream still contains significant locality, though it varies from application to application; and (d) as we move to wider buses, row access time becomes more prominent, making it important to investigate techniques to exploit the available locality to decrease access time.

1 INTRODUCTION

In response to the growing gap between memory access time and processor speed, DRAM manufacturers have created several new DRAM architectures. This paper presents a simulation-based performance study of a representative group, evaluating each in terms of its effect on total execution time. We simulate the performance of seven DRAM architectures: Fast Page Mode [35], Extended Data Out [16], Synchronous [17], Enhanced Synchronous [10], Synchronous Link [38], Rambus [31], and Direct Rambus [32]. While there are a number of academic proposals for new DRAM designs, space limits us to covering only existent commercial parts. To obtain accurate memory-request timing for an aggressive out-of-order processor, we integrate our code into the SimpleScalar tool set [4].

This paper presents a baseline study of a *small-system DRAM organization*: these are systems with only a handful of DRAM chips (0.1–1GB). We do not consider large-system DRAM organizations with many gigabytes of storage that are highly interleaved. The study asks and answers the following questions:

- What is the effect of improvements in DRAM technology on the memory latency and bandwidth problems?

Contemporary techniques for improving processor performance and tolerating memory latency are exacerbating the memory bandwidth problem [5]. Our results show that current DRAM architectures are attacking exactly this problem: the most recent technologies (SDRAM, ESDRAM, and Rambus) have reduced the stall time due to limited bandwidth by a factor of three compared to earlier DRAM architectures. However, the memory-latency component of overhead has not improved.

- Where is time spent in the primary memory system (the memory system beyond the cache hierarchy, but not including secondary [disk] or tertiary [backup] storage)? What is the performance benefit of exploiting the page mode of contemporary DRAMs?

For the newer DRAM designs, the time to extract the required data from the sense amps/row caches for transmission on the memory bus is the largest component in the average access time, though page mode allows this to be overlapped with column access and the time to transmit the data over the memory bus.

- How much locality is there in the address stream that reaches the primary memory system?

The stream of addresses that miss the L2 cache contains a significant amount of locality, as measured by the hit-rates in the DRAM row buffers. The hit rates for the applications studied range 8–95%, with a mean hit rate of 40% for a 1MB L2 cache. (This does not include hits to the row buffers when making multiple DRAM requests to read one cache-line.)

We also make several observations. First, there is a one-time trade-off between cost, bandwidth, and latency: to a point, latency can be decreased by ganging together multiple DRAMs into a wide structure. This trades dollars for bandwidth that reduces latency because a request size is typically much larger than the DRAM transfer width. Page mode and interleaving are similar optimizations that work because a request size is typically larger than the bus width. However, the latency benefits are limited by bus and DRAM speeds: to get further improvements, one must run the DRAM core and bus at faster speeds. Current memory busses are adequate for small systems but are likely inadequate for large ones. Embedded DRAM [5, 19, 37] is not a near-term solution, as its performance is poor on high-end workloads [3]. Faster buses are more likely solutions—witness the elimination of the slow intermediate memory bus in future systems [12]. Another solution is to internally bank the memory array into many small arrays so that each can be accessed very quickly, as in the MoSys Multibank DRAM architecture [39].

Second, widening buses will present new optimization opportunities. Each application exhibits a different degree of locality and therefore benefits from page mode to a different degree. As buses widen, this effect becomes more pronounced, to the extent that different applications can have average access times that differ by 50%. This is a minor issue considering current bus technology. However, future bus technologies will expose the row access as the primary performance bottleneck, justifying the exploration of mechanisms to exploit locality to guarantee hits in the DRAM row buffers: e.g. row-buffer victim caches, prediction mechanisms, etc.

Third, while buses as wide as the L2 cache yield the best memory latency, they cannot halve the latency of a bus half as wide. Page mode overlaps the components of DRAM access when making multiple requests to the same row. If the bus is as wide as a request, one

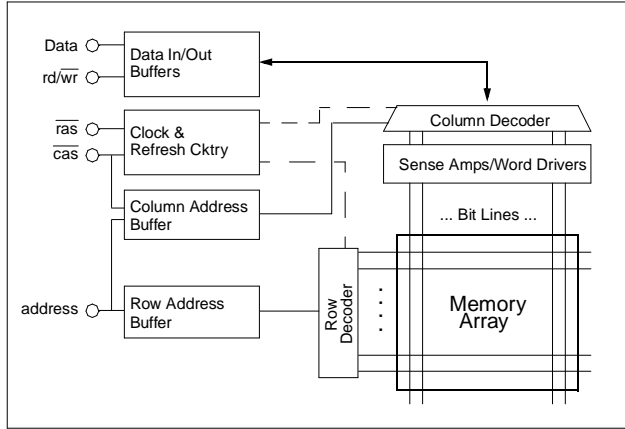


Figure 1: Conventional DRAM block diagram. The conventional DRAM uses a split addressing mechanism still found in most DRAMs today.

cannot exploit this overlap. For cost considerations, having at most an $N/2$ -bit bus, N being the L2 cache width, might be a good choice.

Fourth, critical-word-first does not mix well with burst mode. Critical-word-first is a strategy that requests a block of data potentially out of address-order; burst mode delivers data in a fixed but redefinable order. A burst-mode DRAM can thus have longer latencies in real systems, even if its end-to-end latency is low.

Finally, the choice of refresh mechanism can significantly alter the average memory access time. For some benchmarks and some refresh organizations, the amount of time spent waiting for a DRAM in refresh mode accounted for 50% of the total latency.

As one might expect, our results and conclusions are dependent on our system specifications, which we chose to be representative of mid- to high-end workstations: a 100MHz 128-bit memory bus, an eight-way superscalar out-of-order CPU, lockup-free caches, and a small-system DRAM organization with ~ 10 DRAM chips.

2 RELATED WORK

Burger, Goodman, and Kagi quantified the effect on memory behavior of high-performance latency-reducing or latency-tolerating techniques such as lockup-free caches, out-of-order execution, prefetching, speculative loads, etc. [5]. They concluded that to hide memory latency, these techniques often increase demands on memory bandwidth. They classify memory stall cycles into two types: those due to lack of available memory bandwidth, and those due purely to latency. This is a useful classification, and we use it in our study. This study differs from theirs in that we focus on the access time of only the primary memory system, while their study combines all memory access time, including the L1 and L2 caches. Their study focuses on the behavior of latency-hiding techniques, while this study focuses on the behavior of different DRAM architectures.

Several marketing studies compare the memory latency and bandwidth available from different DRAM architectures [7, 29, 30]. This paper builds on these studies by looking at a larger assortment of DRAM architectures, measuring DRAM impact on total application performance, decomposing the memory access time into different components, and measuring the hit rates in the row buffers.

Finally, there are many studies that measure system-wide performance, including that of the primary memory system [1, 2, 9, 18, 23, 24, 33, 34]. Our results resemble theirs, in that we obtain similar figures for the fraction of time spent in the primary memory system. However, these studies have different goals from ours, in that they are concerned with measuring the effects on total execution time of

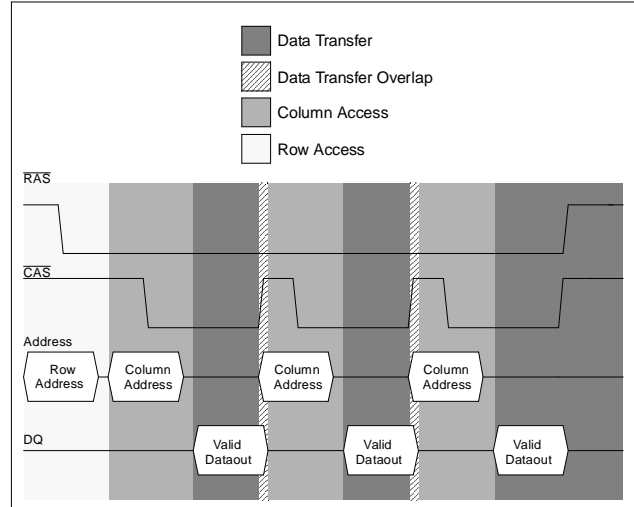


Figure 2: FPM Read Timing. Fast page mode allows the DRAM controller to hold a row constant and receive multiple columns in rapid succession.

varying several CPU-level parameters such as issue width, cache size & organization, number of processors, etc. This study focuses on the performance behavior of different DRAM architectures.

3 BACKGROUND

A Random Access Memory (RAM) that uses a single transistor-capacitor pair for each binary value (bit) is referred to as a Dynamic Random Access Memory or DRAM. This circuit is dynamic because leakage requires that the capacitor be periodically refreshed for information retention. Initially, DRAMs had minimal I/O pin counts because the manufacturing cost was dominated by the number of I/O pins in the package. Due largely to a desire to use standardized parts, the initial constraints limiting the I/O pins have had a long-term effect on DRAM architecture: the address pins for most DRAMs are still multiplexed, potentially limiting performance. As the standard DRAM interface has become a performance bottleneck, a number of “revolutionary” proposals [26] have been made. In most cases, the revolutionary portion is the interface or access mechanism, while the DRAM core remains essentially unchanged.

3.1 The Conventional DRAM

The addressing mechanism of early DRAM architectures is still utilized, with minor changes, in many of the DRAMs produced today. In this interface, shown in Figure 1, the address bus is multiplexed between row and column components. The multiplexed address bus uses two control signals—the row and column address strobe signals, \overline{RAS} and \overline{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.

3.2 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

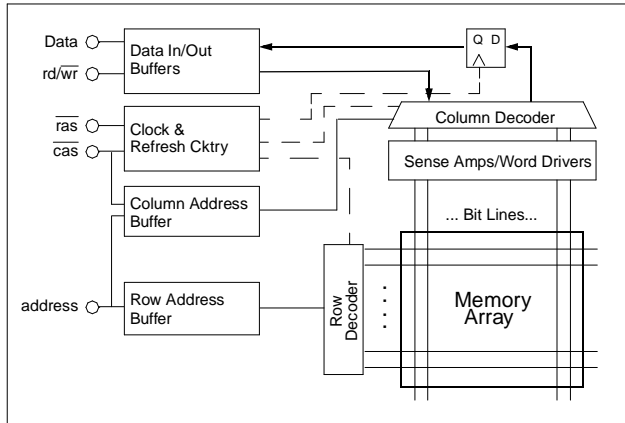


Figure 3: Extended Data Out (EDO) DRAM block diagram. EDO adds a latch on the output that allows CAS to cycle more quickly than in FPM.

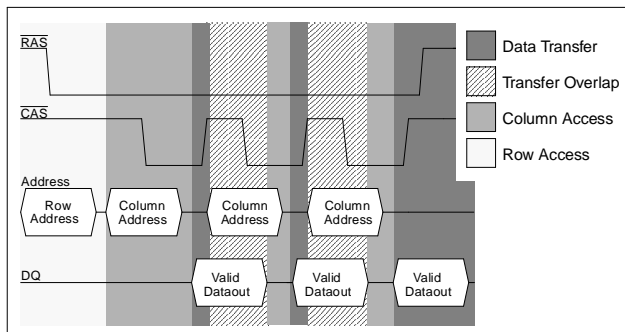


Figure 4: EDO Read Timing. The output latch in EDO DRAM allows more overlap between column access and data transfer than in FPM.

the same row of the DRAM core. Figure 2 gives the timing for FPM reads. The labels show the categories to which the portions of time are assigned in our simulations. Note that page mode is supported in all the DRAM architectures in this study.

3.3 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, shown in Figure 3. 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. Figure 4 gives the timing for an EDO read.

3.4 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. A timing diagram is shown in Figure 5. SDRAM devices typically have a programmable register that holds a bytes-per-request value. SDRAM may therefore return many bytes over several cycles per request. 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.

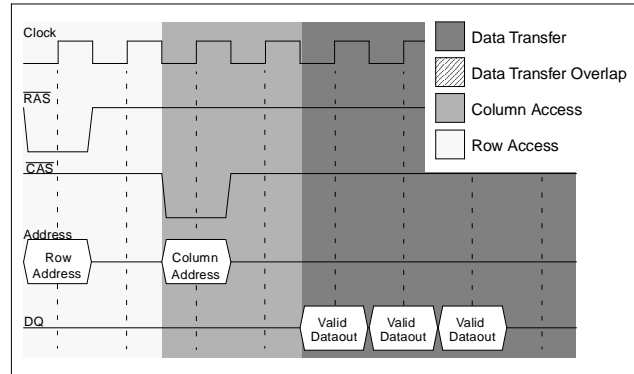


Figure 5: SDRAM Read Operation Clock Diagram. SDRAM contains a writable register for the request length, allowing high-speed column access.

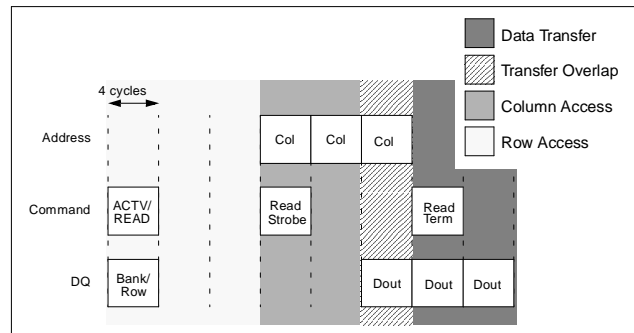


Figure 6: Rambus DRAM Read Operation. Rambus DRAMs transfer on both edges of a fast clock and can handle multiple simultaneous requests.

3.5 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.

3.6 Synchronous Link DRAM (SLDRAM)

RamLink is the IEEE standard (P1596.4) for a bus architecture for devices. Synchronous Link (SLDRAM) is an adaptation of RamLink for DRAM, and is another IEEE standard (P1596.7). Both are adaptations of the Scalable Coherent Interface (SCI). The SLDRAM specification is therefore an open standard allowing for use by vendors without licensing fees. SLDRAM uses a packet-based split request/response protocol. Its bus interface is designed to run at clock speeds of 200-600 MHz and has a two-byte-wide datapath. SLDRAM supports multiple concurrent transactions, provided all transactions reference unique internal banks. The 64Mbit SLDRAM devices contain 8 banks per device.

3.7 Rambus DRAMs (RDRAM)

Rambus DRAMs use a one-byte-wide multiplexed address/data bus to connect the memory controller to the RDRAM devices. The bus runs at 300 Mhz and transfers on both clock edges to achieve a theoretical peak of 600 Mbytes/s. Physically, each 64-Mbit RDRAM is

Table 1: DRAM Specifications used in simulations

DRAM type	Size	Rows	Columns	Transfer Width	Row Buffer	Internal Banks	Speed	Pre-charge	Row Access	Column Access	Data Transfer
FPMDRAM	64Mbit	4096	1024	16 bits	16K bits	1	–	40ns	15ns	30ns	15ns
EDODRAM	64Mbit	4096	1024	16 bits	16K bits	1	–	40ns	12ns	30ns	15ns
SDRAM	64Mbit	4096	256	16 bits	4K bits	4	100MHz	20ns	30ns	30ns	10ns
ESDRAM	64Mbit	4096	256	16 bits	4K bits	4	100MHz	20ns	20ns	20ns	10ns
SLDRAM	64Mbit	1024	128	64 bits	8K bits	8	200MHz	30ns	40ns	40ns	10ns
RDRAM	64Mbit	1024	256	64 bits	16K bits	4	300MHz	26.66ns	40ns	23.33ns	13.33ns
DRDRAM	64Mbit	512	64	128 bits	4K bits	16	400MHz	20/40ns	17.5ns	30ns	10ns

Table 2: Time components in primary memory system

Component	Description
Row Access Time	The time to (possibly) precharge the row buffers, present the row address, latch the row address, and read the data from the memory array into the sense amps
Column Access Time	The time to present the column address at the address pins and latch the value
Data Transfer Time	The time to transfer the data from the sense amps through the column muxes to the data-out pins
Data Transfer Time Overlap	The amount of time spent performing both column access and data transfer simultaneously (when using page mode, a column access can overlap with the previous data transfer for the same row) Note that, since determining the amount of overlap between column address and data transfer can be tricky in the interleaved examples, for those cases we simply call all time between the start of the first data transfer and the termination of the last column access <i>Data Transfer Time Overlap</i> (see Figure 8).
Refresh Time	Amount of time spent waiting for a refresh cycle to finish
Bus Wait Time	Amount of time spent waiting to synchronize with the 100MHz memory bus
Bus Transmission Time	The portion of time to transmit a request over the memory bus to & from the DRAM system that is not overlapped with <i>Column Access Time</i> or <i>Data Transfer Time</i>

divided into 4 banks, each with its own row buffer, and hence up to 4 rows remain active or open¹. Transactions occur on the bus using a split request/response protocol. Because the bus is multiplexed between address and data, only one transaction may use the bus during any 4 clock cycle period, referred to as an octcycle. The protocol uses packet transactions; first an address packet is driven, then the data. Different transactions can require different numbers of octcycles, depending on the transaction type, location of the data within the device, number of devices on the channel, etc. Figure 6 gives a timing diagram for a read transaction.

3.8 Direct Rambus (DRDRAM)

Direct Rambus DRAMs use a 400 Mhz 3-byte-wide channel (2 for data, 1 for addresses/commands). Like the Rambus parts, Direct Rambus parts transfer at both clock edges, implying a maximum bandwidth of 1.6 Gbytes/s. DRDRAMs are divided into 16 banks with 17 half-row buffers². Each half-row buffer is shared between adjacent banks, which implies that adjacent banks cannot be active simultaneously. This organization has the result of increasing the row-buffer miss rate as compared to having one open row per bank, but it reduces the cost by reducing the die area occupied by the row

buffers, compared to 16 full row buffers. A critical difference between RDRAM and DRDRAM is that because DRDRAM partitions the bus into different components, three transactions can simultaneously utilize the different portions of the DRDRAM interface.

4 EXPERIMENTAL METHODOLOGY

For accurate timing of memory requests in a dynamically reordered instruction stream, we integrated our code into SimpleScalar, an execution-driven simulator of an aggressive out-of-order processor [4]. We calculate the DRAM access time, much of which is overlapped with instruction execution. To determine the degree of overlap, and to separate out memory stalls due to bandwidth limitations vs. latency limitations, we run two other simulations—one with perfect primary memory (zero access time) and one with a perfect bus (as wide as an L2 cache line). Following the methodology in [5], we partition the total application execution time into three components: T_P , T_L and T_B which correspond to time spent processing, time spent stalling for memory due to latency, and time spent stalling for memory due to limited bandwidth. In this paper, time spent “processing” includes all activity above the primary memory system, i.e. it contains all processor execution time and L1 and L2 cache activity. Let T be the total execution time for the realistic simulation; let T_U be the execution time assuming unlimited bandwidth—the results from the simulation that models cacheline-wide buses. Then T_P is the time given by the simulation that models a perfect primary memory system, and we calculate T_L and T_B : $T_L = T_U - T_P$ and $T_B = T - T_U$. In addition, we consider one more component: the degree to which the processor is able to overlap memory access time with processing

1. In this study, we model 64-Mbit Rambus parts, which have 4 banks and 4 open rows. Earlier 16-Mbit Rambus organizations had 2 banks and 2 open pages, and future 256-Mbit organizations may have even more.
2. As with the previous part, we model 64-Mbit Direct Rambus, which has this organization. Future (256-Mbit) organizations may look different.

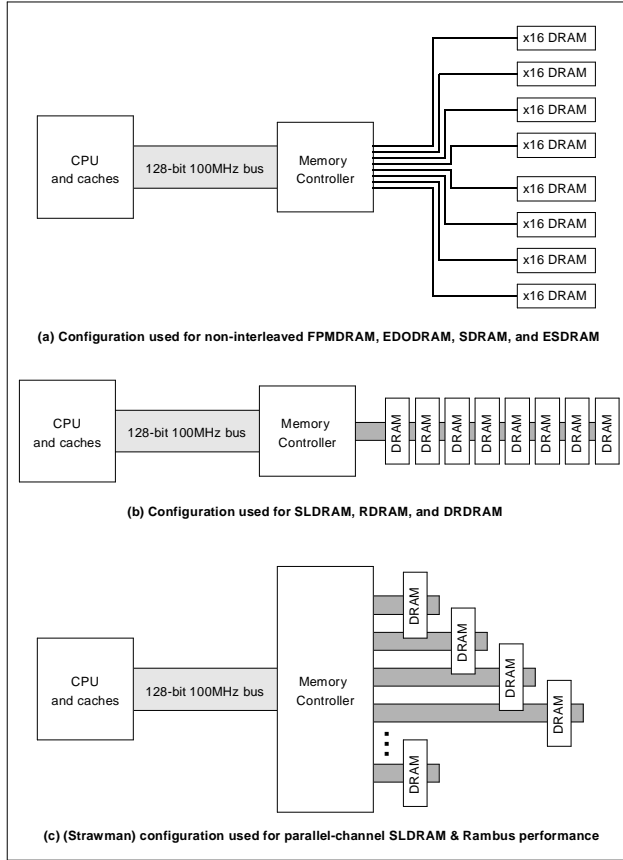


Figure 7: DRAM bus configurations. The DRAM/bus organizations used in (a) the non-interleaved FPM, EDO, SDRAM, and ESDRAM simulations; (b) the SLD and Rambus simulations; and (c) the parallel-channel SLD and Rambus performance numbers in Figure 11. Due to differences in bus design, the only bus overhead included in the simulations is that of the bus that is common to all organizations: the 100MHz 128-bit memory bus.

time. We call this overlapped component T_O , and if T_M is the total time spent in the primary memory system (the time returned by our DRAM simulator), then $T_O = T_P - (T - T_M)$. This is the portion of T_P that is overlapped with memory access.

4.1 DRAM Simulator Overview

The DRAM simulator models the internal state of the following DRAM architectures: Fast Page Mode [35], Extended Data Out [16], Synchronous [17], Enhanced Synchronous [10, 17], Synchronous Link [38], Rambus [31], and Direct Rambus [32].

The timing parameters for the different DRAM architectures are given in Table 1. Since we could not find a 64Mbit part specification for ESDRAM, we extrapolated based on the most recent SDRAM and ESDRAM datasheets. To measure DRAM behavior in systems of differing performance, we varied the speed at which requests arrive at the DRAM. We ran the L2 cache at speeds of 100ns, 10ns, and 1ns, and for each L2 access-time we scaled the main processor’s speed accordingly (the CPU runs at 10x the L2 cache speed).

We wanted a model of a typical workstation, so the processor is eight-way superscalar, out-of-order, with lockup-free L1 caches. L1 caches are split 64KB/64KB, 2-way set associative, with 64-byte linesizes. The L2 cache is unified 1MB, 4-way set associative, write-back, and has a 128-byte linesize. The L2 cache is lockup-free but only allows one outstanding DRAM request at a time; note this orga-

nization fails to take advantage of some of the newer DRAM parts that can handle multiple concurrent requests. 100MHz 128-bit buses are common for high-end machines, so this is the bus configuration that we model. We assume that the communication overhead is only one 10ns cycle in each direction.

The DRAM/bus configurations simulated are shown in Figure 7. For DRAMs other than Rambus and SLD, eight DRAMs are arranged in parallel in a DIMM-like organization to obtain a 128-bit bus. We assume that the memory controller has no overhead delay. SLD, R, and DR utilize narrower, but higher speed busses. These DRAM architectures can be arranged in parallel channels, but we study them here in the context of a single-width DRAM bus, which is the simplest configuration. This yields some latency penalties for these architectures, as our simulations require that the controller coalesce bus packets into 128-bit chunks to be transmitted over the 100MHz 128-bit memory bus. To put the designs on even footing, we ignore the transmission time over the narrow DRAM channel. Because of this organization, transfer rate comparisons may also be deceptive, as we are transferring data from eight conventional DRAM (FPM, EDO, SDRAM, ESDRAM) concurrently, versus only a single device in the case of the narrow bus architectures (SLD, R, DR).

The simulator models a synchronous memory interface: the processor’s interface to the memory controller has a clock signal. This is typically simpler to implement and debug than a fully asynchronous interface. If the processor executes at a faster clock rate than the memory bus (as is likely), the processor may have to stall for several cycles to synchronize with the bus before transmitting the request. We account for the number of stall cycles in *Bus Wait Time*.

The simulator models several different refresh organizations, as described in Section 5. The amount of time (on average) spent stalling due to a memory reference arriving during a refresh cycle is accounted for in the time component labeled *Refresh Time*.

4.2 Interleaving

For the 100MHz 128-bit bus configuration, the transfer size is eight times the request size; therefore each DRAM access is a pipelined operation that takes advantage of page mode. For the faster DRAM parts, this pipeline keeps the memory bus completely occupied. However, for the slower DRAM parts (FPM and EDO), the timing looks like that shown in Figure 8(a). While the address bus may be fully occupied, the memory bus is not, which puts the slower DRAMs at a disadvantage compared to the faster parts. For comparison, we model the FPM and EDO parts as interleaved as well (shown in Figure 8(b)). The degree of interleaving is that required to occupy the memory data bus as fully as possible. This may actually over-occupy the address bus, in which case we assume that there are more than one address buses between the controller and the DRAM parts. FPM DRAM specifies a 40ns CAS period and is four-way interleaved; EDO DRAM specifies a 25ns CAS period and is two-way interleaved. Both are interleaved at a bus-width granularity.

5 EXPERIMENTAL RESULTS

For most graphs, the performance of several DRAM organizations is given: FPM1, FPM2, FPM3, EDO1, EDO2, SDRAM, ESDRAM, SLD, R, and DR. The first two configurations (FPM1 and FPM2) show the difference between always keeping the row buffer open (thereby avoiding a precharge overhead if the next access is to the same row) and never keeping the row buffer open. FPM1 is the pessimistic strategy of closing the row buffer after every access and precharging immediately; FPM2 is the optimistic strategy of keeping the row buffer open and delaying precharge. The dif-

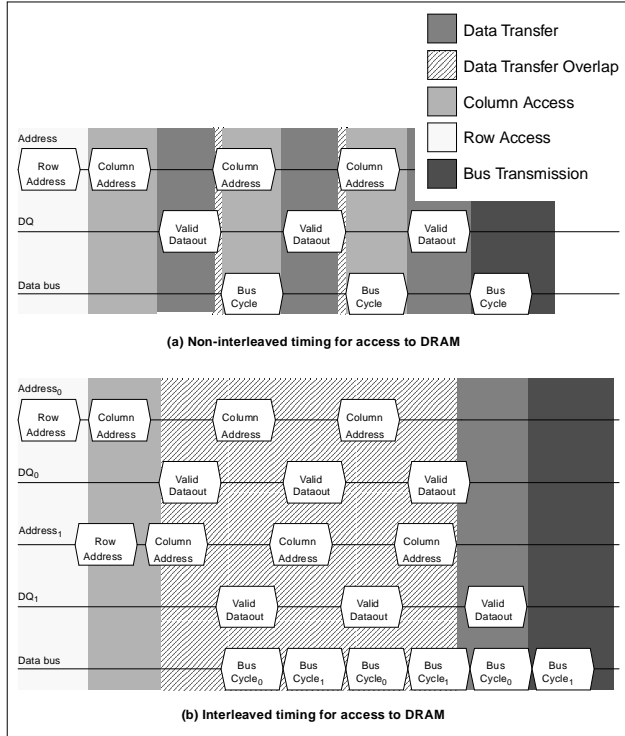


Figure 8: Interleaving in DRAM simulator. Time in *Data Transfer Overlap* accounts for much activity in interleaved organizations; *Bus Transmission* is the remainder of time that is not overlapped with anything else.

ference is seen in *Row Access Time*, which, as the graphs show, is not large for present-day organizations. For all other DRAM simulations but ESDRAM, we keep the row buffer open, as the timing of the pessimistic strategy can be calculated without simulation. The FPM3 and EDO2 labels represent the interleaved organizations of FPM and EDO DRAM. The remaining labels are self-explanatory.

5.1 Handling Refresh

Surprisingly, DRAM refresh organization can affect performance dramatically. Where the refresh organization is not specified for an architecture, we simulate a model in which the DRAM allocates bandwidth to either memory references or refresh operations, at the expense of predictability [26]. The refresh period for all DRAM parts but Rambus is 64ms; Rambus parts have a refresh period of 33ms. In the simulations presented in this paper, this period is divided into N individual refresh operations that occur $33/N$ milliseconds apart, where 33 is the refresh period in milliseconds and N is the number of rows in an internal bank times the number of internal banks. This is the Rambus mechanism, and a memory request can be delayed at most the refresh of one DRAM row. For Rambus parts, this behavior is spelled out in the data sheets. For other DRAMs, the refresh mechanism is not explicitly stated. Note that normally, when multiple DRAMs are ganged together into physical banks, all banks are refreshed at the same time. This is different; Rambus refreshes internal banks individually.

Because many textbooks describe the refresh operation as a periodic shutting down of the DRAM until all rows are refreshed (e.g. [14]), we also simulated stalling the DRAM once every 64ms to refresh the entire memory array; thus, every 64ms, one can potentially delay one or more memory references the time it takes to refresh the entire memory array. This approach yields refresh stalls

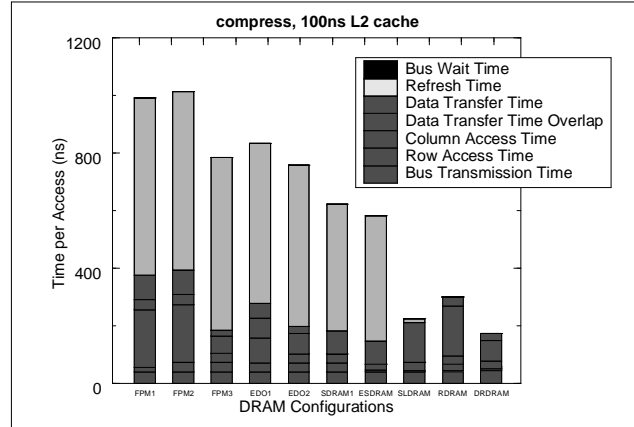


Figure 9: The penalty for choosing the wrong refresh organization. In some instances, time waiting for refresh can account for more than 50%.

up to two orders of magnitude worse than the time-interspersed scheme. Particularly hard-hit was the *compress* benchmark, shown in Figure 9. Because such high overheads are easily avoided with an appropriate refresh organization, we only present results for the time-interspersed refresh approach.

5.2 Total Execution Time

Figure 10(a) shows the total execution time for several benchmarks of SPECint '95 using SDRAM for the primary memory system. The time is divided into processor computation, which includes accesses to the L1 and L2 caches, and time spent in the primary memory system. The graphs also show the overlap between processor computation and DRAM access time. For each architecture, there are three vertical bars, representing L2 cache cycle times of 100ns, 10ns, and 1ns (left, middle, and rightmost bars, respectively). For each DRAM architecture and L2 cache access time, the figure shows a bar representing execution time, partitioned into four components:

- Memory stall cycles due to limited bandwidth
- Memory stall cycles due to latency
- Processor time (includes L1 and L2 activity) that is overlapped with memory access
- Processor time (includes L1 and L2 activity) that is **not** overlapped with memory access

SimpleScalar schedules instructions extremely aggressively and hides much of the memory latency with other work—though this “other work” is not all useful work, as it includes all L1 and L2 cache activity. For the 100ns L2 (corresponding to a 100MHz processor), between 50% and 99% of the memory access-time is hidden, depending on the type of DRAM the CPU is attached to (the faster DRAM parts allow a processor to exploit greater degrees of concurrency). For 10ns (corresponding to a 1GHz processor), between 5% and 90% of the latency is hidden. As expected, the slower systems hide more of the DRAM access time than the faster systems.

Figure 10(b) shows that the more advanced DRAM designs have reduced the proportion of overhead attributed to limited bandwidth by roughly a factor of three: from 3 CPI in FPMDRAM to 1 CPI in SDRAM, ESDRAM, and DRDRAM.

Summary: The graphs demonstrate the degree to which contemporary DRAM designs are addressing the memory bandwidth problem. Popular high-performance techniques such as lockup-free

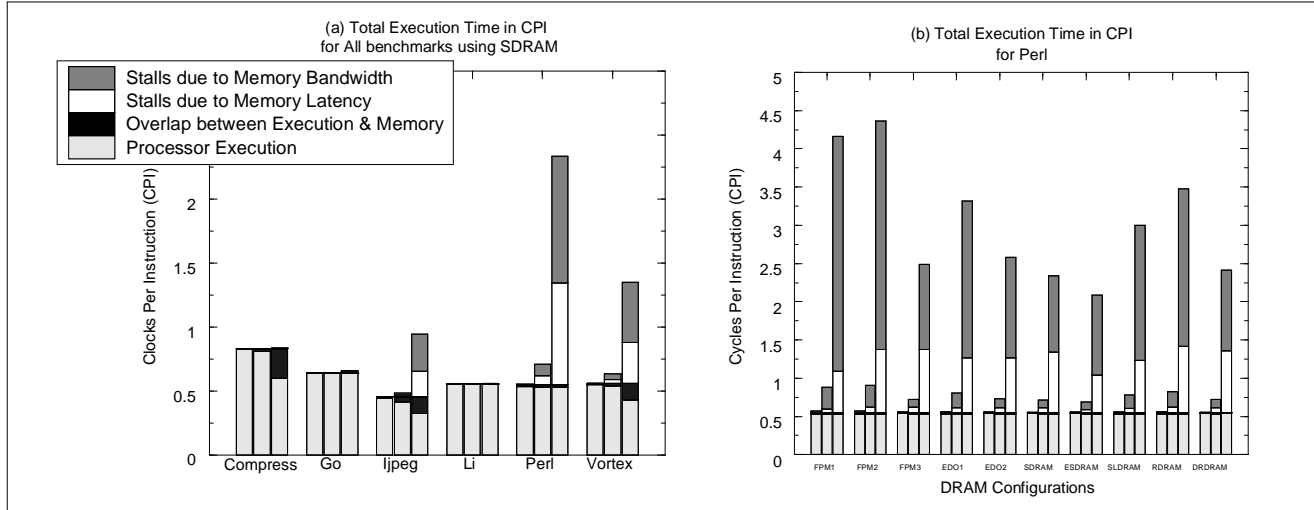


Figure 10: Total execution time + access time to the primary memory system. Figure (a) gives total execution time in units of CPI for different DRAM types. The overhead is broken into processor time and memory time, with overlap between the two shown, and memory cycles are divided into those due to limited bandwidth and those due to latency. Figure (b) shows the total execution time in CPI for all benchmarks, using Synchronous DRAM.

caches and out-of-order execution expose memory bandwidth as the bottleneck to improving system performance; i.e., common techniques for improving CPU performance and tolerating memory latency are exacerbating the memory bandwidth problem [5]. Our results show that contemporary DRAM architectures are attacking exactly that problem. We see that the most recent technologies (SDRAM, ESDRAM, SLDRAM, and Rambus designs) have reduced the stall time due to limited bandwidth by a factor of two to three, as compared to earlier DRAM architectures. Unfortunately, there are no matching improvements in memory latency; while the newest generation of DRAM architectures decreases the cost of limited bandwidth by a factor of three compared to the previous generation, the cost of stalls due to latency has remained almost constant.

The graphs also show the expected result that as L2 cache and processor speeds increase, systems are less able to tolerate memory latency. Accordingly, the remainder of our study focuses on the components of memory latency.

5.3 Average Memory Latency

Figure 11 breaks down the memory-system component of Figure 10. The access times are divided by the number of accesses to obtain an average time-per-DRAM-access. This is end-to-end latency: the time to complete an entire request, as opposed to critical-word latency. Much of this time is overlapped with processor execution; the degree of overlap depends on the speed of the L2 cache and main CPU. Since the variations in performance are not large, we only show three benchmarks that vary most widely. The differences are almost entirely due to *Row Access Time* and *Bus Transmission Time*.

Row Access Time varies with the hit rate in the row buffers, which, as later graphs show, is as application-dependent as cache hit rate. The pessimistic FPM1 strategy of always closing pages wins out over the optimistic FPM2 strategy. However, with larger caches, we have seen many instances where the open-page strategy wins; compulsory DRAM accesses tend to exhibit good locality.

The differences between benchmarks in *Bus Transmission Time* are due to write traffic. Writes allow a different degree of overlap between the column access, data transfer, and bus transmission. The heavier the write traffic, the higher the *Bus Transmission* component. One can conclude *perl* and *jpeg* have heavier write traffic than *go*.

Though it is a completely unbalanced design, we also measured

latencies for 128-bit wide configurations for Rambus and SLDRAM designs, pictured in Figure 7(c). These “parallel-channel” results are intended to demonstrate the mismatch between today’s bus speeds and fastest DRAMs; they are shown in the bottom right of Figure 11.

Bus Transmission Time is that portion of the bus activity not overlapped with column access or data transfer, and it accounts for 10% to 30% of the total latency. In the parallel-channel results, it accounts for more than 50%. This suggests that, for some DRAM architectures, bus speed is becoming a critical issue. While current technologies seem balanced, bus speed is likely to become a significant problem very quickly for next-generation DRAMs [8]. It is interesting to note that the recently announced Alpha 21364 integrates Rambus memory controllers onto the CPU and connects the processor directly to the DRDRAMs with a 400MHz Rambus Channel, thereby eliminating the slow intermediate bus [12].

EDO DRAM does a much better job than FPM DRAM of overlapping column access with data transfer. This is to be expected, given the timing diagrams for these architectures. Note that the overlap components (*Data Transfer Time Overlap*) tend to be very large in general, demonstrating relatively significant performance savings due to page-mode. This is an argument for keeping buses no wider than half the block size of the L2 cache.

Several of the architectures show no overlap at all between data transfer and column access. SDRAM and ESDRAM do not allow such overlap because they instead use burst mode, which obviates multiple column accesses (see Figure 5). SLDRAM does allow overlap, just as the Rambus parts do; however, for simplicity, in our simulations we modeled SLDRAM’s burst mode. The overlapped mode would have yielded similar latencies.

The interleaved configurations (FPM3 and EDO2) demonstrate excellent performance; latency for FPM DRAM improves by a factor of 2 with four-way interleaving, and EDO improves by 25-30% with two-way interleaving. The interleaved EDO configuration performs slightly worse than the FPM configuration because it does not take full advantage of the memory bus; there is still a small amount of unused bus bandwidth. Note that the break-downs of these organizations look very much like Direct Rambus; Rambus behaves similarly to highly interleaved systems but at much lower cost points.

The time stalled due to refresh tends to account for 1-2% of the total latency; this is more in line with expectations than the results shown in Figure 9. The time stalled synchronizing with the memory

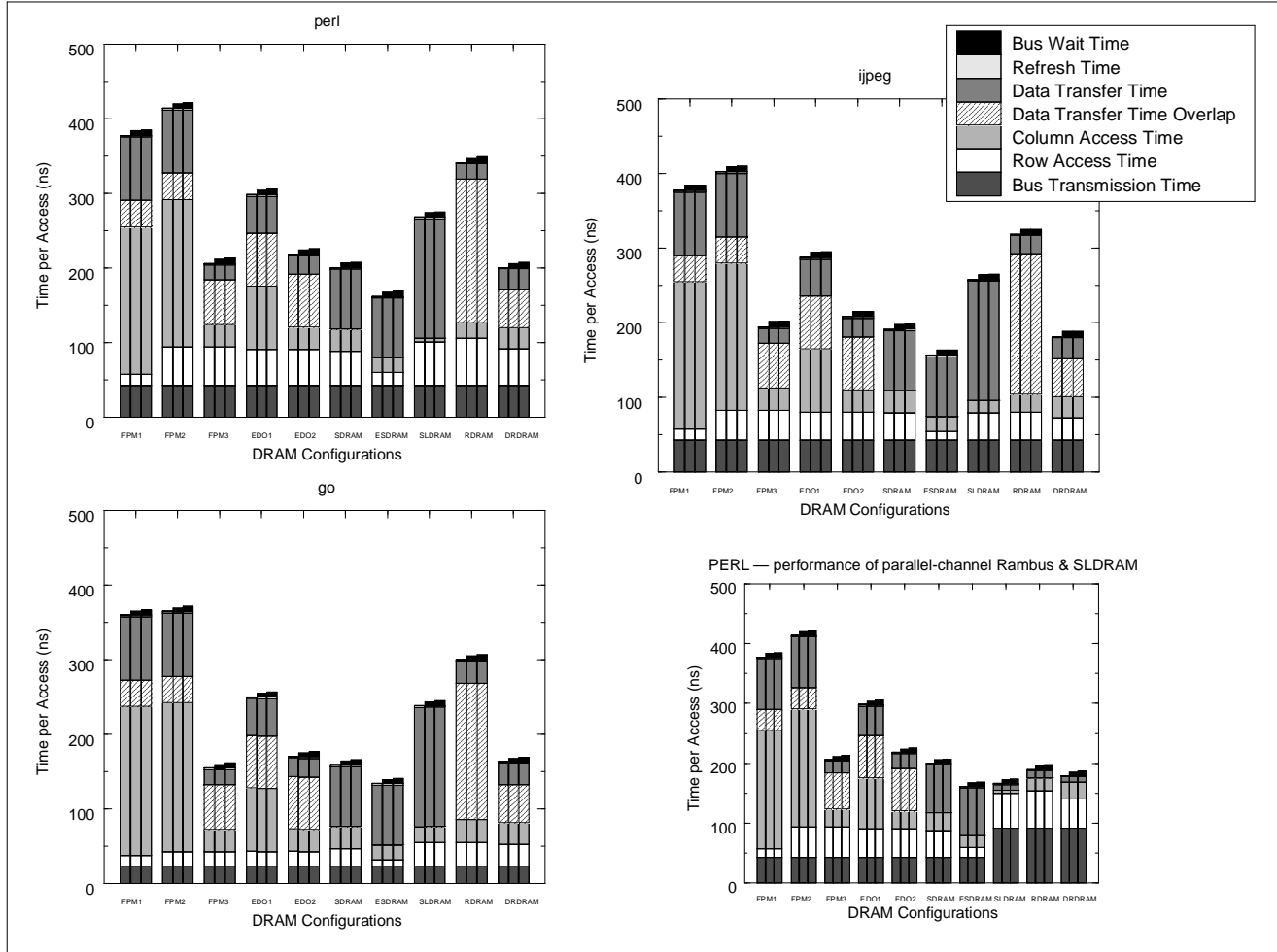


Figure 11: Break-downs for primary memory access time, 128-BIT BUS. These graphs present the average access time on a 128-bit bus across DRAM architectures for the three benchmarks that display the most widely varying behavior. The different DRAM architectures display significantly different access times. The main cause for variation from benchmark to benchmark is the *Row Access Time*, which varies with the probability of hitting an open page in the DRAM's row buffers. If a benchmark exhibits a high degree of locality in its post-L2 address stream, it will tend to have a small *Row Access Time* component.

bus is in the same range, accounting for 1-5% of the total. This is a small price to pay for a simpler DRAM interface, compared to a fully asynchronous design.

Summary: The FPM architecture is the baseline architecture, but it could be sped up by 30% with a greater degree of overlap between the column access and data transmission. This is seen in the EDO architecture: its column access is a bit faster due to the latch between the sense amps and the output pins, and its degree of overlap with data transfer is greater, yielding a significantly faster design using essentially the same technology as FPM. Synchronous DRAM is another 30% faster than EDO, and Enhanced SDRAM increases performance another 15% by improving the row- and column-access timing parameters and adding an SRAM cache to improve concurrency, though we note that its improvement over SDRAM is less dramatic if using a pessimistic close-page strategy as in FPM1.

As modeled, SLDRAM and Rambus designs have higher end-to-end transaction latencies than SDRAM or ESDRAM, as they require multiple narrow-bus cycles to complete a 128-bit transaction. However, they are not ganged together into a wide datapath, as are the other organizations. Despite the handicap, SLDRAM performs well, which is important considering it is a public standard. Direct Rambus comes out about equal to SDRAM in end-to-end latency. Note that SLDRAM and RDRAM make twice the number of data

transfers as DRDRAM; had they been organized as two DRAMs wide to put them on equal footing with DRDRAM, which has a 128-bit transfer width, their latencies would be 20-30% lower.

Where the SLDRAM and Rambus designs excel is in their critical-word latencies: though SDRAM and ESDRAM win in end-to-end latency, they are rigid in their access ordering. Parts like Rambus and SLDRAM are like the interleaved FPM and EDO organizations in that they allow the memory controller to request the components of a large block in arbitrary order. Thus, the Rambus parts allow easy critical-word-first ordering, whereas burst-mode DRAMs do not.

Last, the parallel-channel results demonstrate the failure of a 100MHz 128-bit bus to keep up with today's fastest parts. Here, we have placed enough channels side-by-side to create a 128-bit datapath that is then pushed across the 100MHz bus, and Direct Rambus has roughly the same end-to-end latency as before. Clearly, we are pushing the limits of today's busses. The Alpha 21364 will solve this problem by ganging together multiple Rambus Channels connected directly to the CPU, eliminating the 100MHz bus [12].

5.4 Cost-Performance Considerations

The organizations are equal in their capacity: all but the interleaved examples use eight 64Mbit DRAMs. The FPM3 organization uses

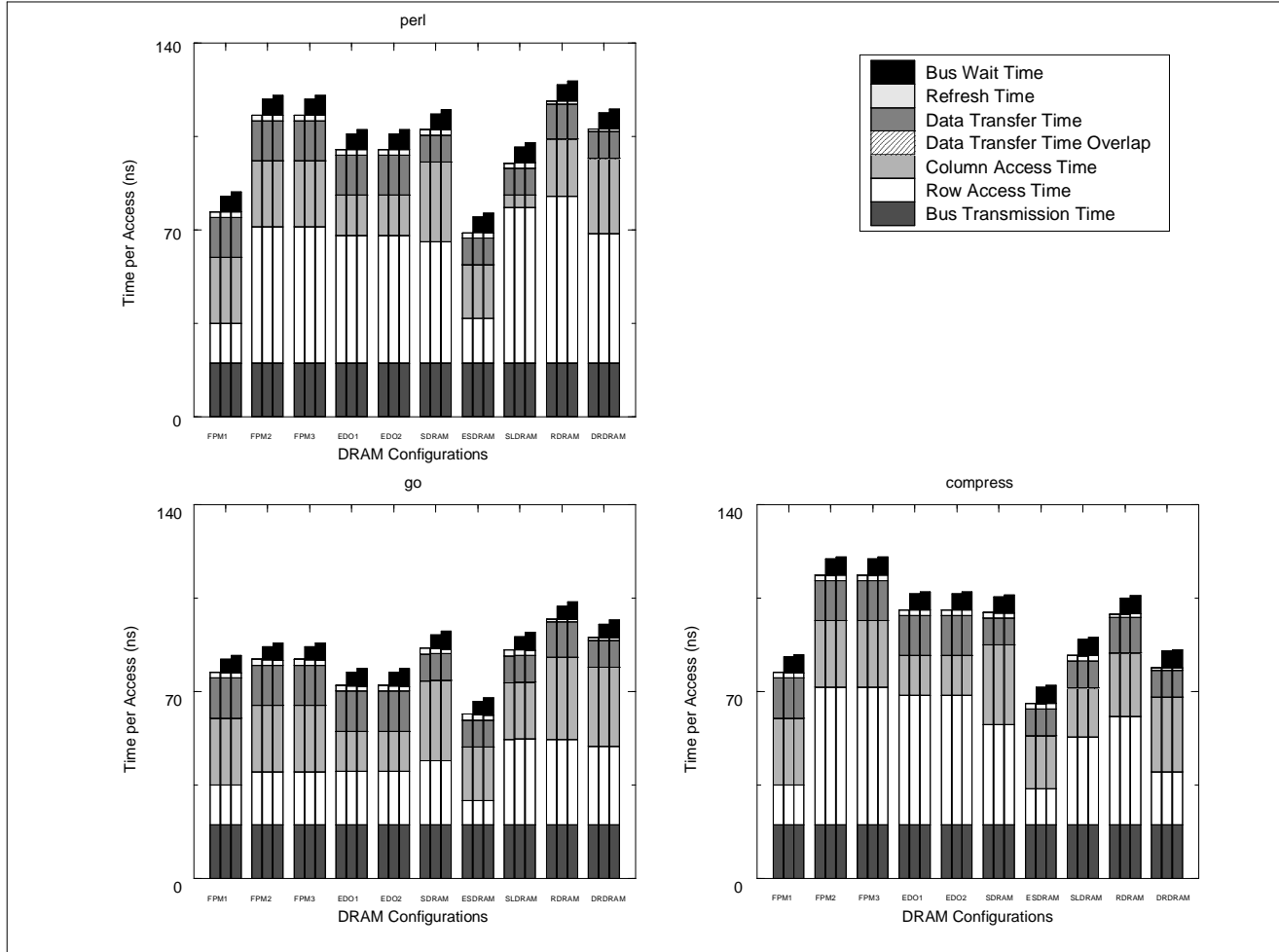


Figure 12: Break-downs for primary memory access time, 128-BYTE BUS. These graphs present the average access time on a 128-byte bus, the same width as an L2 cache line. Therefore the pipelined access to memory (multiple column accesses per row access) is not seen, and the *Row Access* component becomes relatively more significant than in the results of a 128-bit bus (Figure 11). Whereas in Figure 11, variations in *Row Access* caused overall variations in access time of roughly 10%, these graphs quantify the effect that *Row Access* has on systems with wider buses: average access time can vary by a factor of two.

32 64Mbit DRAMs, and the EDO2 organization uses sixteen. However, the cost of each system is very different. Cost is a criterion in DRAM selection that may be as important as performance. Each of these DRAM technologies carries a different price, and these prices are dynamic, based on factors including number of suppliers, sales volume, die area premium, and speed yield.

In the narrow-bus organization we modeled, money spent on Rambus and SLDRAM parts does not go directly to latency's bottom line as with the other DRAMs. The average access time graphs demonstrate how effectively dollars reduce latency: the only reason FPM, EDO, SDRAM, and ESDRAM have latencies comparable to Rambus and SLDRAM is that they are ganged together into very wide organizations that deliver 128 bits of data per request, though each individual DRAM transfers only 16 bits at a time. If each organization had been represented by a single 64Mbit DRAM, the FPM, EDO, SDRAM, and ESDRAM parts would have had latencies from four to eight times those in Figure 11. The Rambus and SLDRAM parts benefit by multiple DRAMs only in that this organization extends the size of the collective sense-amp cache and thus increases the row-buffer hit rates (see Figure 13); a single Rambus or SLDRAM chip will perform almost as well as a group of eight.

Ignoring price premiums, cost is a good argument for the high-speed narrow-bus DRAMs. Rambus and SLDRAM parts give the

performance of other DRAM organizations at a fraction of the cost (roughly 1/32 the interleaved FPM organization, 1/16 the interleaved EDO organization, and 1/8 all the non-interleaved organizations). Alternatively, by ganging together several Rambus Channels, one can achieve better performance at the same cost. Accordingly, these parts typically carry a stiff price premium, but typically less than 8x.

5.5 Perfect-Width Buses

As a limit study, we measured the performance of a perfect-width bus: 100MHz and as wide as an L2 cache line. The results are shown in Figure 12. The scale is much smaller than the previous graphs, and some components have scaled with the change in bus width. The number of column accesses are reduced by a factor of eight, which reduces the *Column Access* and *Data Transfer* times. The row access remains the same, as does *Bus Wait Time*; they appear to have increased in importance. Bus transmission for a read has been reduced from 90ns (10 for the request, 80 to transmit the data), much of which was overlapped with column access and data transfer, to 20ns, none of which is overlapped. Because each request requires only one memory access, there is no pipelining to be exploited, and the full 20ns transmission is exposed (10ns each for address and data). FPM2 and FPM3 look identical, as do EDO1 and EDO2. This

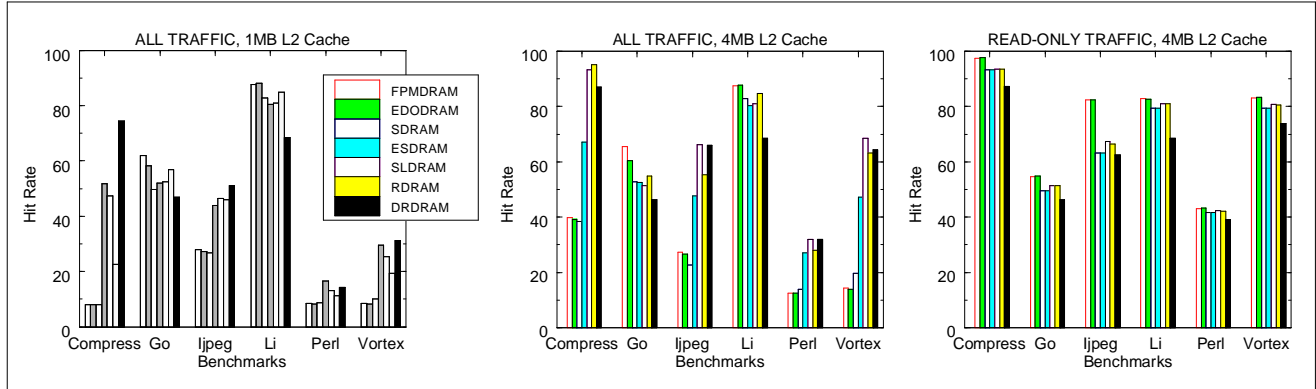


Figure 13: Hit-rates in the row buffers. These graphs show hit-rates for the benchmarks on each of the DRAM architectures. The newer DRAMs, with more internal banking, tend to have higher hit rates. Write traffic, due to writebacks, disrupts the locality of the address stream for architectures with fewer internal banks.

is no mistake. Two configurations are interleaved, the others are not. Making the bus the width of the request size obviates interleaving.

There are no *Overlap* components in these graphs. With a 128-byte bus, each cache line fill requires a single transaction. Overlap is possible if multiple concurrent requests to the DRAM are allowed, but this is beyond the scope of our current DRAM simulations. Overlap shown in previous graphs is due to the overlap of multiple requests required for a single cache line fill.

As before, the primary variation between benchmarks is the *Row Access Time*. The variations are larger than in the previous graphs, because the row access time is proportionally much larger. The graphs show that the locality of reference for each application (seen in the row-buffer hit-rates, Figure 13) can have a dramatic impact on the access latency—for example, there is a factor of two difference between the average access latency for *compress* and *perl*. This effect has been seen before—McKee’s work shows that intentionally reordering memory accesses to exploit locality can have an order of magnitude effect on memory-system performance [21, 22].

Summary: Coupled with extremely wide buses that hide the effects of limited bandwidth and thus highlight the differences in memory latency, the DRAM architectures perform similarly. As FPM1 and ESDRAM show, the variations in *Row Access* can be avoided by always closing the row buffer after an access and hiding the sense-amp precharge time during idle moments. This yields the best measured performance and its performance is much more deterministic (e.g. FPM1 yields the same *Row Access* independent of benchmark). Note that in studies with a 4MB L2 cache, some benchmarks executing with an optimistic strategy showed very high row-buffer hit rates and had *Row Access* components that were near-zero.

Comparing these results to the previous experiment, we see that when one considers current technology (128-bit buses), there is little variation from application to application in the average memory access time. The two components that vary, *Row Access* and *Bus Transmission*, contribute little to the total latency, being overshadowed by long memory-access pipelines that exploit page mode. However, moving to wider buses decreases the column accesses per request, and as a result the row access, which is much larger than column access to begin with, becomes significant. With fewer column accesses per request, we are less able to hide bus transmission time, and this component becomes more noticeable as well.

Variations in row access time, though problematic for real-time systems, do offer an opportunity to optimize performance: one can easily imagine enhanced row-buffer caching schemes, row-buffer victim caches, or even prediction mechanisms that attempt to capitalize on the amount of post-L2-cache locality. However, with current organizations, such measures make little sense.

5.6 Row-Buffer Hit Rates

Associated with each DRAM core is a set of sense amps that can latch data; this essentially amounts to an SRAM cache, and internally-banked DRAMs have several of these caches. Collectively, a DRAM or bank of DRAMs can have a sizable SRAM cache (call it a *row-buffer cache*) in these sense amps. The size of each DRAM’s row-buffer cache is the product of the *Row Buffer* and *Internal Banks* terms in Table 1—except for DRDRAM, which has 17 half-row buffers shared between 16 banks (a total of 68K bits of storage).

Figure 13 presents the variations in hit rates for the row-buffer caches of different DRAM architectures. Hit rate does **not** include the effect of hits that are due to multiple requests to satisfy one L2 cacheline: these results are for the 128-byte bus. We present results for two types of traffic: all traffic and read-only traffic. The read-only results ignore all writes to the DRAM system (which, in write-back caches, occur only when a line is being replaced).

The results show that memory requests frequently hit the row buffers; for the full-traffic simulations hit rates range from 8–95%, with a mean of 40% (mean calculated over only the 1MB L2 cache results). There is a significant change in hit rate when writes are included in the address stream: including write traffic tends to decrease the row-buffer hit-rate for those DRAMs with less SRAM storage. Writebacks tend to purge useful data from the smaller row-buffer caches; thus the Rambus, SLDRAM, and ESDRAM parts perform better than the others. This effect suggests that when writebacks happen, they do so without much locality: the cachelines that are written back tend to be to DRAM pages that have not been accessed recently. This is expected behavior.

Note that a designer can play with the ordering of address bits to maximize the row-buffer hits. A similar technique is used in interleaved memory systems to obtain the highest bandwidth.

5.7 Trace-Driven Simulations

We also investigated the effect of using trace-driven simulation to measure memory latency. We simulated the same benchmarks using SimpleScalar’s in-order mode with single-issue. Clearly, in-order execution cannot yield the same degree of overlap as out-of-order execution, but we did see virtually identical average access times compared to out-of-order execution, for both 128-bit and 128-byte buses. Because SPEC has been criticized as being not representative of real-world applications, we also used University of Washington’s Etch traces [11] to corroborate what we had seen using SPEC on SimpleScalar. The Etch benchmarks yielded very similar results, with the main difference being that the row-buffers had a higher

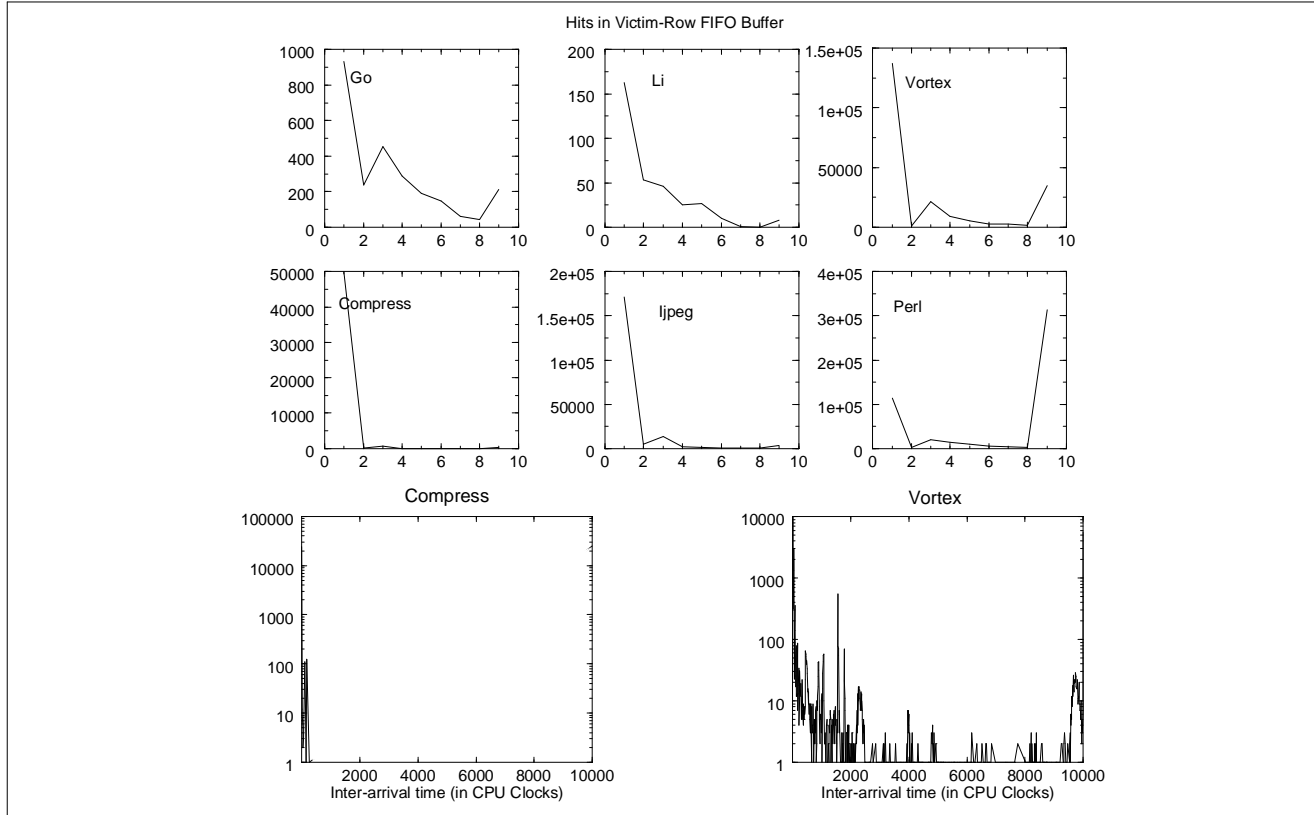


Figure 14: Locality in the stream of accesses to the single open row in the FPM DRAM. The top six graphs show the frequency with which accesses to a given DRAM page hit at stack depth x . The bottom two graphs show the inter-arrival time of accesses that hit in an open DRAM page. Both sets of graphs show that when references are made to the data in a particular DRAM page, the accesses tend to be localized in time.

average hit rate (61% with a 4MB L2 cache, as opposed to 51%), and a smaller standard deviation (11.4, as opposed to 25.5).

6 CONCLUSIONS

We have simulated seven commercial DRAM architectures in a workstation-class setting, connected to a fast, out-of-order, eight-way superscalar processor with lockup-free caches. We have found the following: (a) contemporary DRAM technologies are addressing the memory bandwidth problem but not the memory latency problem; (b) the memory latency problem is closely tied to current mid-to high-performance memory bus speeds (100MHz), which will soon become inadequate for high-performance DRAM designs; (c) there is a significant degree of locality in the addresses that are presented to the primary memory system—this locality seems to be exploited well by DRAM designs that are multi-banked internally and therefore have more than one row buffer; and (d) exploiting this locality will become more important in future systems when memory buses widen, exposing row access time as a significant factor.

The bottom line is that contemporary DRAM architectures have used page-mode and internal interleaving to achieve a one-time performance boost. These techniques improve bandwidth directly and improve latency indirectly by pipelining over the memory bus the multiple transactions that satisfy one read or write request (requests are often cacheline-sized, and the cache width is typically greater than the bus width). This is similar to the performance optimization of placing multiple DRAMs in parallel to achieve a bus-width datapath: this optimization works because the bus width is typically greater than an individual DRAM’s transfer width. We have seen

that each of the DRAM architectures studied takes advantage of internal interleaving and page mode to differing degrees of success. However, as the studies show, we will soon hit the limit of these benefits: the limiting factors are now the speed of the bus and, to a lesser degree, the speed of the DRAM core. To improve performance further, we must explore other avenues.

7 FUTURE WORK

We will extend the research to cover large systems, which have different performance behavior. In the present study, the number of DRAMs per organization is small, therefore the hit rate seen in the row buffers can be high. In larger systems, this effect decreases in significance. For instance, in large systems, bandwidth is more of an issue than latency—hitting an open page is less important than scheduling the DRAM requests so as to avoid bus conflicts.

As buses grow wider, *Row Access Time* becomes significant; in our 1MB L2 studies it accounts for 20–50% of the total latency, and in our 4MB L2 studies it accounted for 1–50% [8]. Increasing the number of open rows is one approach to decreasing the overhead, as seen in the multi-banked DRAMs such as Rambus and SLDRAM. Other approaches include adding extra row buffers to cache previously opened rows, prefetching into the row buffers, placing row-buffer victim-caches onto the chips, predicting whether or not to close an open page, etc. We intend to look into this more closely, but wanted to get a rough idea of the potential gains. We kept the last eight accessed row buffers in a FIFO and kept track of the number of hits and misses to the buffer, as well as the depth at which any hits occurred. The results are shown in Figure 14. For each benchmark,

we show the number of **misses** to the **main row buffer**. The first value at the leftmost of each curve is the number of hits at a depth of one in the FIFO victim buffer. The next value represents the number of hits at a depth of two, and so on. The rightmost value in each curve is the number of accesses that missed both the main row buffer and the FIFO victim buffer. The two graphs on the bottom show the amount of locality in the two benchmarks with the most widely varying behavior; the graphs plot the time in CPU clocks between successive references to the previous open row (i.e. the row that was replaced by the currently open row: it also happens to be the topmost entry in the FIFO). This graph demonstrates that when the row is accessed in the future, it is most often accessed in the very near future. Our conclusion is that the previously-referenced row has a high hit rate, and it is likely to be referenced within a short period of time if it is referenced again at all. A number of proven techniques exist to exploit this behavior, such as victim caching, set associative row buffers, etc.

ACKNOWLEDGMENTS

This study grew out of research begun by Brian Davis and extended by Vinodh Cuppu, Özkan Dikmen, and Rohit Grover in a graduate-level architecture class taught by Prof. Jacob in the spring of 1998. Dikmen and Grover were instrumental in the development of the simulator used in this study.

We would like to thank several researchers at IBM who provided helpful insight into the internal workings of the various DRAM architectures: Mark Charney, Paul Coteus, Phil Emma, Jude Rivers, and Jim Rogers. We would also like to thank Sally McKee for her detailed comments on and suggestions for the paper, as well as the anonymous reviewers of the first draft.

Trevor Mudge is supported in part by DARPA grant DABT 63-96-C0047. Vinodh Cuppu and Bruce Jacob are supported in part by NSF grant EIA-9806645.

REFERENCES

- [1] L. A. Barroso, et al. "Memory system characterization of commercial workloads." In *Proc. ISCA-25*, June 1998, pp. 3–14.
- [2] D. Bhandarkar and J. Ding. "Performance characterization of the Pentium Pro processor." In *Proc. HPCA-3*, February 1997, pp. 288–297.
- [3] N. Bowman, et al. "Evaluation of existing architectures in IRAM systems." *Workshop on Mixing Logic and DRAM*, June 1997.
- [4] D. Burger and T. M. Austin. "The SimpleScalar tool set, version 2.0." Tech. Rep. CS-1342, University of Wisconsin-Madison, June 1997.
- [5] D. Burger, et al. "Memory bandwidth limitations of future microprocessors." In *Proc. ISCA-23*, May 1996, pp. 78–89.
- [6] M. Charney, P. Coteus, P. Emma, J. Rivers, and J. Rogers. *Private communication*. 1999.
- [7] R. Crisp. "Direct Rambus technology: The new main memory standard." *IEEE Micro*, vol. 17, no. 6, pp. 18–28, November 1997.
- [8] V. Cuppu and B. Jacob. "The performance of next-generation DRAM architectures." Tech. Rep. UMD-SCA-TR-1999-1, University of Maryland Systems and Computer Architecture Group, March 1999.
- [9] Z. Cvetanovic and D. Bhandarkar. "Characterization of Alpha AXP performance using TP and SPEC workloads." In *Proc. ISCA-21*, April 1994, pp. 60–70.
- [10] ESDRAM. *Enhanced SDRAM 1M x 16*. Enhanced Memory Systems, http://www.edram.com/products/datasheets/16M_esdram0298a.pdf, 1998.
- [11] Etch. *Memory System Research at the University of Washington*. The University of Washington, <http://etch.cs.washington.edu/>, 1998.
- [12] L. Gwennap. "Alpha 21364 to ease memory bottleneck." *Microprocessor Report*, vol. 12, no. 14, pp. 12–15, October 1998.
- [13] L. Gwennap. "New processor paradigm: V-IRAM." *Microprocessor Report*, vol. 12, no. 3, pp. 17–19, March 1998.
- [14] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, 2nd Ed.* Morgan Kaufmann Publishers, Inc., 1996.
- [15] S. I. Hong, et al. "Access order and effective bandwidth for streams on a Direct Rambus memory." *Proc. HPCA-5*, January 1999, pp. 80–89.
- [16] IBM. *EDO DRAM 4M x 16 Part No. IBM0165165PT3C*. <http://www.chips.ibm.com/products/memory/88H2011/88H2011.pdf>, 1998.
- [17] IBM. *SDRAM 1M x 16 x 4 Bank Part No. IBM0364164*. <http://www.chips.ibm.com/products/memory/19L3265/19L3265.pdf>, 1998.
- [18] K. Keeton, et al. "Performance characterization of a quad Pentium Pro SMP using OLTP workloads." *Proc. ISCA-25*, June 1998, pp. 15–26.
- [19] C. Kozyrakis, et al. "Scalable processors in the billion-transistor era: IRAM." *IEEE Computer*, vol. 30, no. 9, pp. 75–78, September 1997.
- [20] D. Kroft. "Lockup-free instruction fetch/prefetch cache organization." In *Proc. ISCA-8*, May 1981.
- [21] S. McKee, et al. "Design and evaluation of dynamic access ordering hardware." In *Proc. International Conference on Supercomputing*, May 1996.
- [22] S. A. McKee and W. A. Wulf. "Access ordering and memory-conscious cache utilization." *Proc. HPCA-1*, January 1995, pp. 253–262.
- [23] B. Nayfeh, et al. "Evaluation of design alternatives for a multiprocessor microprocessor." In *Proc. ISCA-23*, May 1996, pp. 67–77.
- [24] B. A. Nayfeh, et al. "The impact of shared-cache clustering in small-scale shared-memory multiprocessors." *HPCA-2*, Feb. 96, pp. 74–84.
- [25] Y. Nunomura, et al. "M32R/D—integrating DRAM and microprocessor." *IEEE Micro*, vol. 17, no. 6, pp. 40–48, Nov. 1997.
- [26] S. Przybylski. *New DRAM Technologies: A Comprehensive Analysis of the New Architectures*. MicroDesign Resources, Sebastopol CA, 1996.
- [27] Rambus. "Rambus memory: Enabling technology for PC graphics." Tech. Rep., Rambus Inc., Mountain View CA, October 1994.
- [28] Rambus. "64-megabit Rambus DRAM technology directions." Tech. Rep., Rambus Inc., Mountain View CA, September 1995.
- [29] Rambus. "Comparing RDRAM and SGRAM for 3D applications." Tech. Rep., Rambus Inc., Mountain View CA, October 1996.
- [30] Rambus. "Memory latency comparison." Tech. Rep., Rambus Inc., Mountain View CA, September 1996.
- [31] Rambus. *16/18Mbit & 64/72Mbit Concurrent RDRAM Data Sheet*. Rambus, <http://www.rambus.com/docs/Cnctds.pdf>, 1998.
- [32] Rambus. *Direct RDRAM 64/72-Mbit Data Sheet*. Rambus, <http://www.rambus.com/docs/64dDDS.pdf>, 1998.
- [33] P. Ranganathan, et al. "Performance of database workloads on shared-memory systems with out-of-order processors." In *Proc. ASPLOS-8*, October 1998, pp. 307–318.
- [34] M. Rosenblum, et al. "The impact of architectural trends on operating system performance." In *Proc. SOSP-15*, December 1995.
- [35] Samsung. *FPM DRAM 4M x 16 Part No. KM416V4100C*. Samsung Semiconductor, [http://www.usa.samsungsemi.com/products/prod-spec/dramcomp/KM416V40\(1\)00C.PDF](http://www.usa.samsungsemi.com/products/prod-spec/dramcomp/KM416V40(1)00C.PDF), 1998.
- [36] I. Sase, et al. "Multimedia LSI accelerator with embedded DRAM." *IEEE Micro*, vol. 17, no. 6, pp. 49–54, November 1997.
- [37] A. Saulsbury, et al. "Missing the memory wall: The case for processor/memory integration." In *Proc. ISCA-23*, May 1996, pp. 90–101.
- [38] SDRAM. *4M x 18 SDRAM Advance Datasheet*. SDRAM, Inc., <http://www.sldram.com/Documents/corp400b.pdf>, 1998.
- [39] R. Wilson. "MoSys tries synthetic SRAM." *EE Times Online*, July 15, 1997. <http://www.eetimes.com/news/98/1017news/tries.html>.