

High-Performance DRAMs in Workstation Environments

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

This paper presents a simulation-based performance study of several of the new high-performance DRAM architectures, each evaluated in a small system organization. These small-system organizations correspond to workstation-class computers and use only a handful of DRAM chips (~10, as opposed to ~1 or ~100). The study covers Fast Page Mode, Extended Data Out, Synchronous, Enhanced Synchronous, Dual Data Rate, 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; (d) systems without L2 caches are feasible for low- and medium-speed CPUs (1GHz and below); and (e) 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 [36], Extended Data Out [19], Synchronous [20], Enhanced Synchronous [13], Dual Data Rate [21], Synchronous Link [38], Rambus [32], and Direct Rambus [33]. While there are a number of academic proposals for new DRAM designs, space limits us to covering only existing commercial architectures. 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. We also study a set of benchmarks that are appropriate for such systems: user-class applications such as compilers and small databases rather than server-class applications such as transaction processing systems. 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, DDR, 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 2–97%, 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.)

- Does it make sense to eliminate the L2 cache in low-cost systems?

Modern DRAM designs are increasing the amount of SRAM and other cache-like storage on the DRAM die [12]. In most cases, a memory system comprised of multiple DRAM chips will have many kilobytes of high-speed memory (for example, 8KB of high-speed storage per DRAM is common today, and the amount is increasing quickly). Our simulations show that for low- and medium-speed

CPUs (1GHz and under), it is possible to eliminate the L2 cache and still have very reasonable performance.

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. One can essentially pay for bandwidth and simultaneously reduce latency, as a request size is typically much larger than the DRAM transfer width, and the increased bandwidth improves the transfer time of the large request. Both page mode and interleaving exploit this phenomenon. However, once the bus is as wide as a fundamental data unit, the benefit diminishes, and to obtain further improvements, one must run the DRAM core and bus at faster speeds. Though current memory buses are adequate for current low- to mid-end systems, they are inadequate for high-end systems. Wider busses via embedded DRAM [5, 23, 37] are not a near-term solution, as embedded DRAM 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 [16]. 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 a factor of two. This is a minor issue considering current bus technology. However, future bus technologies will expose the row access as a primary performance bottleneck, justifying the exploration of mechanisms that exploit locality to guarantee hits in the DRAM row buffers: e.g. row-buffer victim caches, prediction mechanisms, etc. Note that recent commercial DRAM proposals address exactly this issue by placing associative SRAM caches on the DRAM die to exploit locality and the tremendous bandwidth available on-chip [12].

Third, while buses as wide as the L2 cache yield the best memory latency, they have passed the point of diminishing returns: for instance, a bus half as wide would not yield twice the latency. The use of page mode overlaps the components of DRAM access when making multiple requests to the same row, and one can only exploit this overlap when a cache block is larger than the bus width—otherwise, every cache-fill request requires one row access and one column access. Therefore, the DRAM bus should not exceed $N/2$ bits, where N is the L2 cache width.

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. However, we note that for the applications studied, total execution time seems to correlate more with end-to-end DRAM latencies than with critical-word latencies.

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 organization that is found in SPARC workstations and has the same bandwidth as a DRDRAM channel), 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 the 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 [6, 30, 31]. 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, 10, 22, 26, 27, 34, 35]. 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 varying several

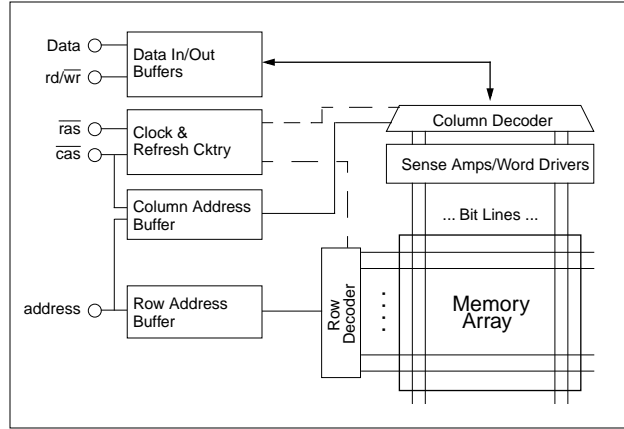


Figure 1: Conventional DRAM block diagram. The conventional DRAM uses a split addressing mechanism still found in most DRAMs today.

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 [28] 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, $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ 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

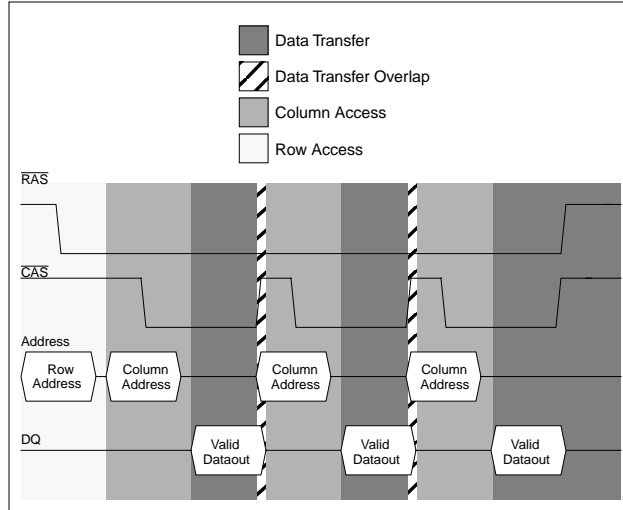


Figure 2: FPM Read Timing. Fast page mode allows the DRAM controller to hold a row constant and receive multiple columns in rapid succession.

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 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 investigated 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 $\overline{\text{CAS}}$ 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.

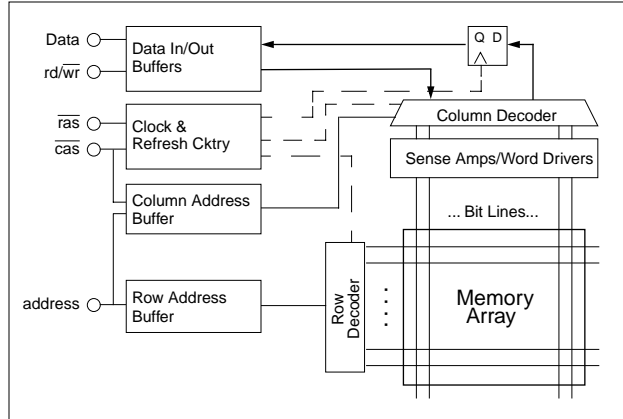


Figure 3: Extended Data Out (EDO) DRAM block diagram. EDO adds a latch on the output that allows CAS to cycle more quickly than in FPM.

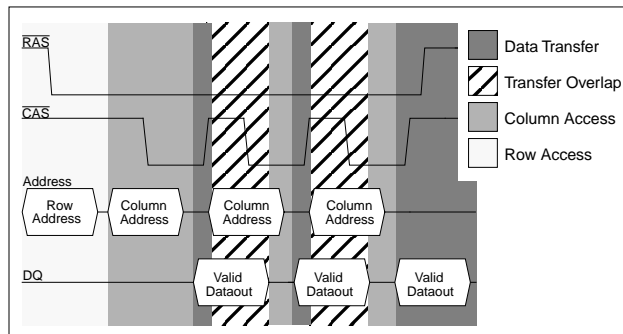


Figure 4: EDO Read Timing. The output latch in EDO DRAM allows more overlap between column access and data transfer than in FPM.

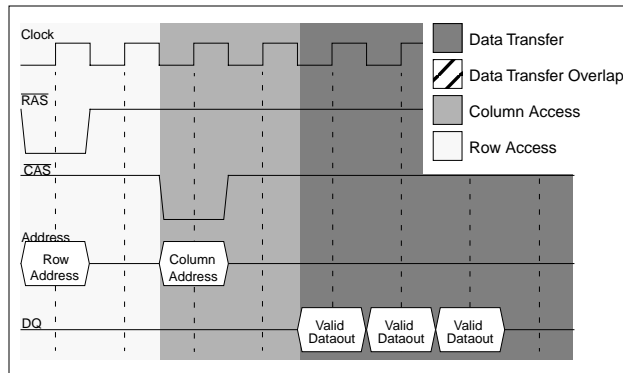


Figure 5: SDRAM Read Operation Clock Diagram. SDRAM contains a writable register for the request length, allowing high-speed column access.

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 burst length or 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.

3.5 Enhanced Synchronous DRAM (ESDRAM)

Enhanced Synchronous DRAM is a 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 inter-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. It also allows a write to proceed through the sense amps directly without overwriting the line buffered in the SRAM cache, which would otherwise destroy any read locality.

3.6 Double Data Rate DRAM (DDR DRAM)

Double data rate (DDR) DRAM doubles the bandwidth available from SDRAM by transferring data at both edges of the clock. DDR DRAM are very similar to single data rate SDRAM in all other characteristics. They use the same signalling technology, the same interface specification, and the same pinouts on the DIMM carriers. Internally, DDR-DRAM employs $2n$ prefetching, where twice the number of bits is read in or written to the DRAM array on each access, and two n -bit transfers take place every half cycle.

3.7 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.

Note that SLDRAM is currently only of academic interest; the SLDRAM standards development effort has recently been abandoned, and it is unlikely that any SLDRAM chips will ever be produced.

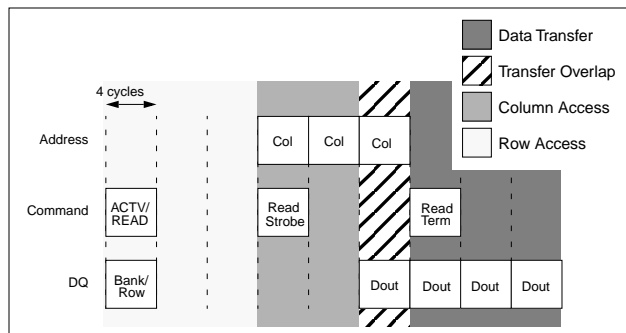


Figure 6: Rambus DRAM Read Operation. Rambus DRAMs transfer on both edges of a fast clock and can handle multiple simultaneous requests.

3.8 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 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.

3.9 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

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.

bus into different components, three transactions can simultaneously utilize the different portions of the DRDRAM interface.

4 EXPERIMENTAL METHODOLOGY

To obtain 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 memory stalls due to bandwidth limitations from memory stalls due to 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, respectively, 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_R 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 can calculate T_L and T_B as follows: $T_L = T_U - T_P$ and $T_B = T_R - T_U$. In addition, we consider the degree to which the processor is successful in overlapping memory access time with processing time. We call the overlap 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 $T_O = T_P - (T_R - T_M)$. This is the portion of T_P that is overlapped with memory access.

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
DDR	128Mbit	4096	512	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	20ns	17.5ns	30ns	10ns

4.1 DRAM Simulator Overview

The DRAM simulator models the internal state of the following DRAM architectures: Fast Page Mode [36], Extended Data Out [19], Synchronous [20], Enhanced Synchronous [13, 20], Dual Data Rate [21], Synchronous Link [38], Rambus [32], and Direct Rambus [33].

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).

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>

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, writeback, and has a 128-byte linesize. The L2 cache is lockup-free but only allows one outstanding DRAM request at a time; note this organization fails to take advantage of some of the newer DRAM parts that can handle multiple concurrent requests. However, this is addressed later on in the discussion and in two follow-on studies of ours [7, 8, 11]. 100MHz 128-bit buses are common for high-end workstations, so this is the bus configuration that we model. Note that it also has the same bandwidth as Direct Rambus (1.6 GB/s). We assume that the communication overhead is only one 10ns cycle in each direction. For the DDR simulations, the bus transfers data on both edges of the clock; therefore, its effective bandwidth is twice that of the other simulations.

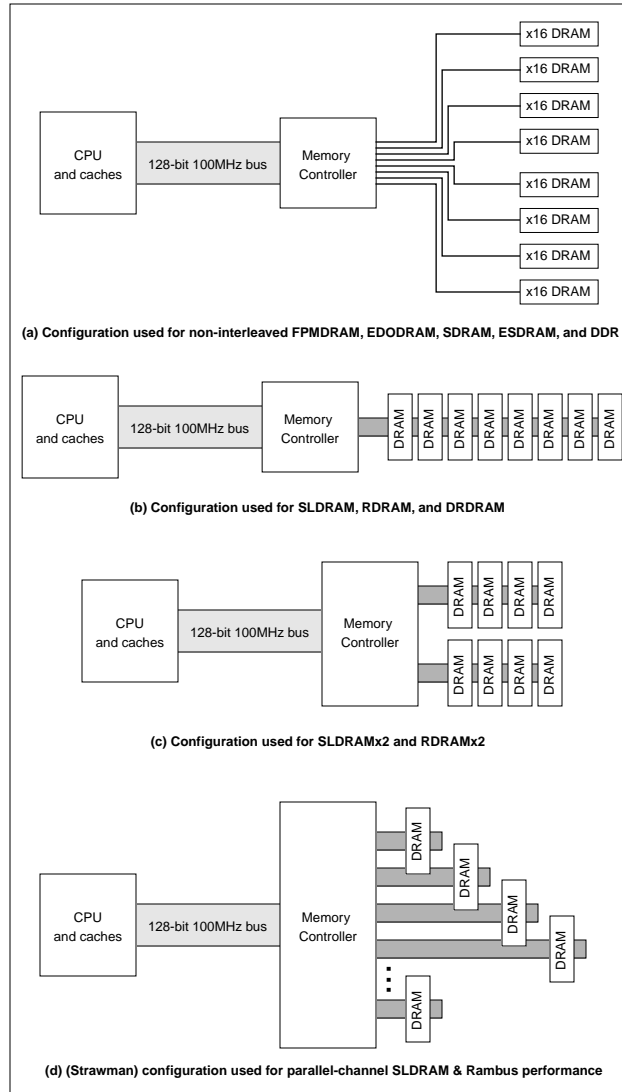


Figure 7: DRAM bus configurations. The DRAM/bus organizations used in (a) the non-interleaved FPM, EDO, SDRAM, and ESDRAM simulations; (b) the SLD and Rambus simulations; (c) the SLD and Rambus dual-channel organizations; and (d) the parallel-channel SLD and Rambus performance numbers in Figure 11. Due 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.

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. SLD, R, and DR utilize narrower, but higher speed buses. These DRAM architectures can be arranged in parallel channels, and we study them here in the context of a single-width DRAM bus, which is the simplest configuration, as well as a dual-channel configuration for SLD and R. As in real-world systems, the memory controller coalesces bus packets into 128-bit chunks to be

transmitted over the 100MHz 128-bit memory bus. To keep the designs on even footing, we ignore the overhead of the memory controller. Because of the narrow-channel organization, transfer rate comparisons may also be deceptive, as we are transferring data from eight conventional DRAM (FPM, EDO, SDRAM, ESDRAM, DDR) concurrently, versus only a single device in the case of the narrow-channel architectures (SLDRAM, RDRAM, DRDRAM).

As mentioned, for SLDRAM and RDRAM we also model two-channel systems to observe their behavior when their bandwidth is equivalent to the other DRAM organizations. The FPM, EDO, SDRAM and ESDRAM organizations connect the DRAMs to the memory controller via a 128-bit 100MHz bus (1.6 GB/s bandwidth). DRDRAM uses a 16-bit 800MHz bus (also 1.6 GB/s bandwidth). However, SLDRAM and RDRAM have native bandwidths of 800 MB/s and 600 MB/s, respectively. We measure the performance of the native bandwidths of SLDRAM and RDRAM, and we also measure the performance of ganged organizations using two buses side-by side, whose aggregate bandwidth is 1.6 GB/s. For comparison, we also look at one of the newer DRAM technologies: 128-bit 100MHz DDR, which has twice the bandwidth of the others: 3.2GB/s.

To better distinguish results from different benchmarks, we do not begin taking measurements (or warming the caches) until the application has finished its initialization stage, during which its memory accesses are extremely sequential in nature. The memory accesses that we see thus tend to better reflect the true behavior of each benchmark.

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

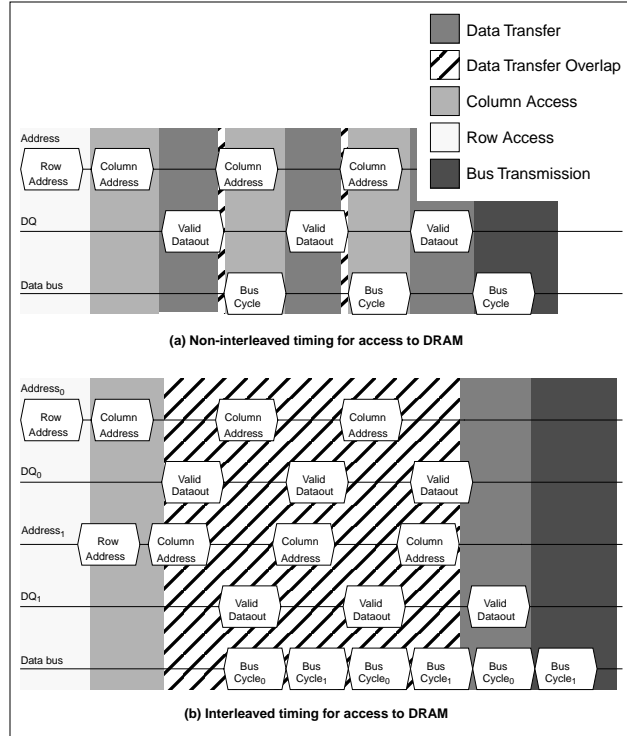


Figure 8: Interleaving in DRAM simulator. Time in *Data Transfer Overlap* accounts for much activity in interleaved organizations; *Bus Transmission* is the remainder of time that is not overlapped with anything else.

EDO), the timing looks like that shown in Figure 8(a). While the address bus may be fully occupied, the memory data 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 in interleaved organizations 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, DDR, SLDRAM, SLDRAMx2, RDRAM, RDRAMx2, and DRDRAM. 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 difference 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 SLD RAMx2 and RDRAMx2 labels represent the SLD RAM and RDRAM organizations with two channels (described earlier). The remaining labels should be 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 [28]. 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. [17]), 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 up to two orders of magnitude worse than the time-interspersed scheme. Particularly hard-hit was the *compress* benchmark, shown in Figure 9 with refresh stalls accounting for over 50% of the average access time in several of the DRAM architectures. Because such high overheads are easily avoided with an appropriate refresh organization, we only present results for the time-interspersed refresh approach.

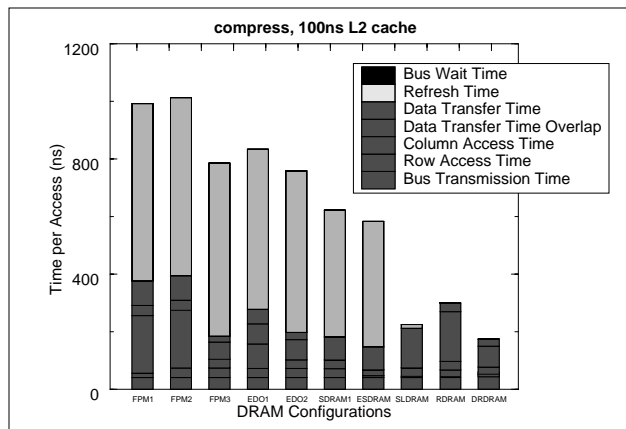


Figure 9: The penalty for choosing the wrong refresh organization. In some instances, time waiting for refresh can account for more than 50%.

5.2 Total Execution Time

Figure 10(a) shows the total execution time for several benchmarks of SPECint '95³ 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

One of the most obvious results is that more than half of the SPECint '95 benchmarks (gcc, jpeg, m88ksim, perl, and vortex) exhibit the same memory-system overhead that has been reported in the literature for large-footprint applications considered much more memory-intensive than SPEC: the middle bars in Figure 10(a) for these benchmarks, which represent CPU speeds of 1GHz, have non-overlapped

3. We do not look at the floating-point benchmarks here because their regular access patterns make them easy targets for optimizations such as prefetching and access reordering [24, 25].

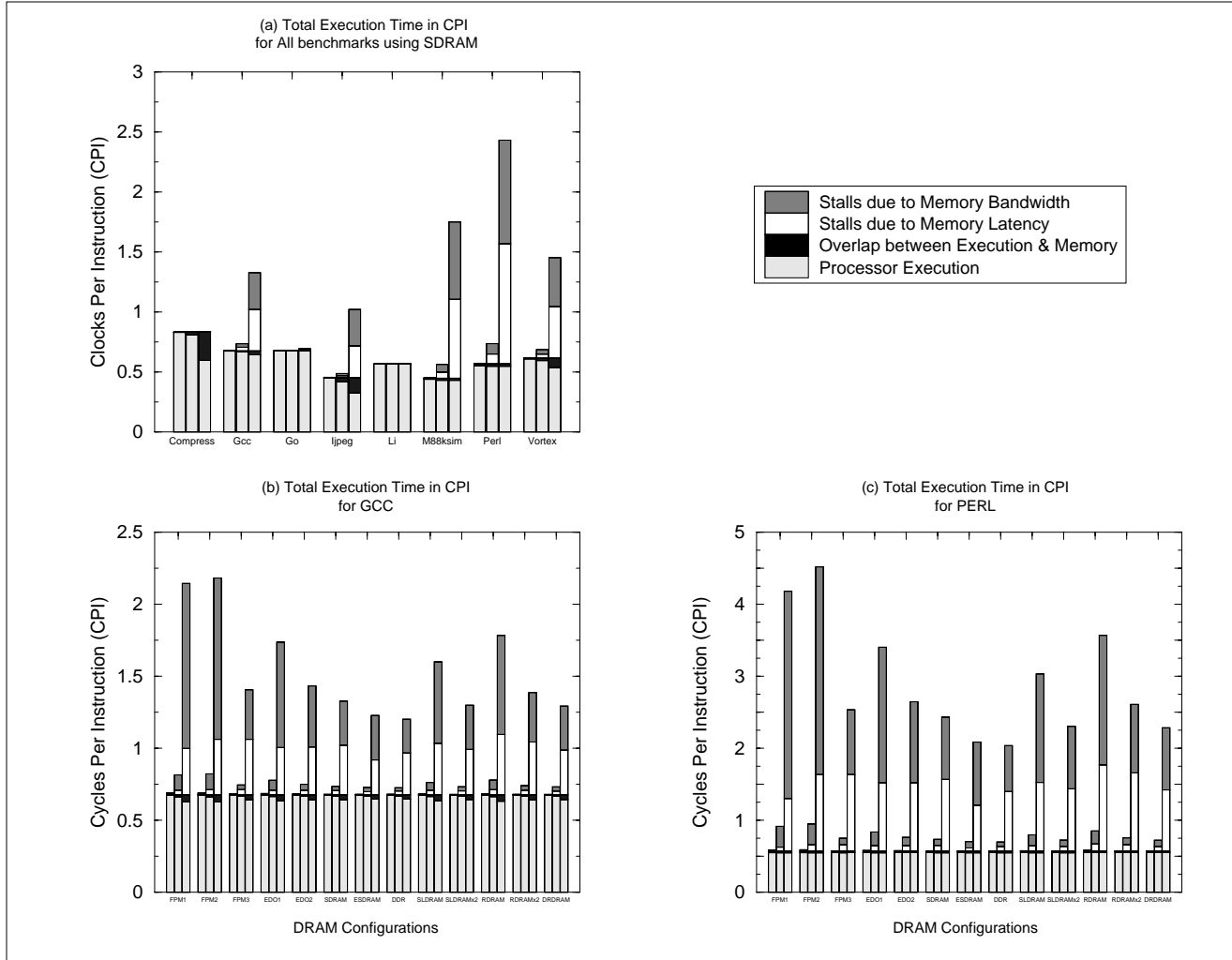


Figure 10: Total execution time + access time to the primary memory system. Figure (a) shows the total execution time in CPI for all benchmarks, using Synchronous DRAM. Figures (b) and (c) give total execution time in units of CPI for different DRAM types. The overhead is broken into processor time and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

DRAM components constituting 10–25% of the total execution time. This echoes published results for DRAM overheads in commercial workloads such as transaction processing [1, 2, 10, 22].

Another obvious point is that anywhere from 5% to 99% of the memory overhead is overlapped with processor execution—the most memory-intensive applications successfully overlap 5–20%. SimpleScalar schedules instructions extremely aggressively and hides a fair amount 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.

Figures 10(b) and 10(c) show that the more advanced DRAM designs have reduced the proportion of overhead attributed to limited bandwidth by roughly a factor of three: e.g., looking at the 10ns bars (corresponding to 10GHz CPUs) for both GCC and PERL benchmarks, the *Stalls Due to Memory Bandwidth* component decreases from 3 for PERL and 1.5 for GCC in the FPMDRAM organization to 1 for PERL and 0.5 for GCC in the SDRAM, ESDRAM, DDR, and DRDRAM organizations.

The figures also show the difference in performance due to DRAM architectures. For today's high-end DRAMs (e.g. SDRAM, Direct Rambus, ESDRAM, and DDR), there is little difference in total execution time. The rankings do not change from application to application (DDR is fastest, followed by ESDRAM, Direct Rambus, and SDRAM), and the gap between the fastest and slowest architectures is only 10–15%.

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 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, DDR, 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

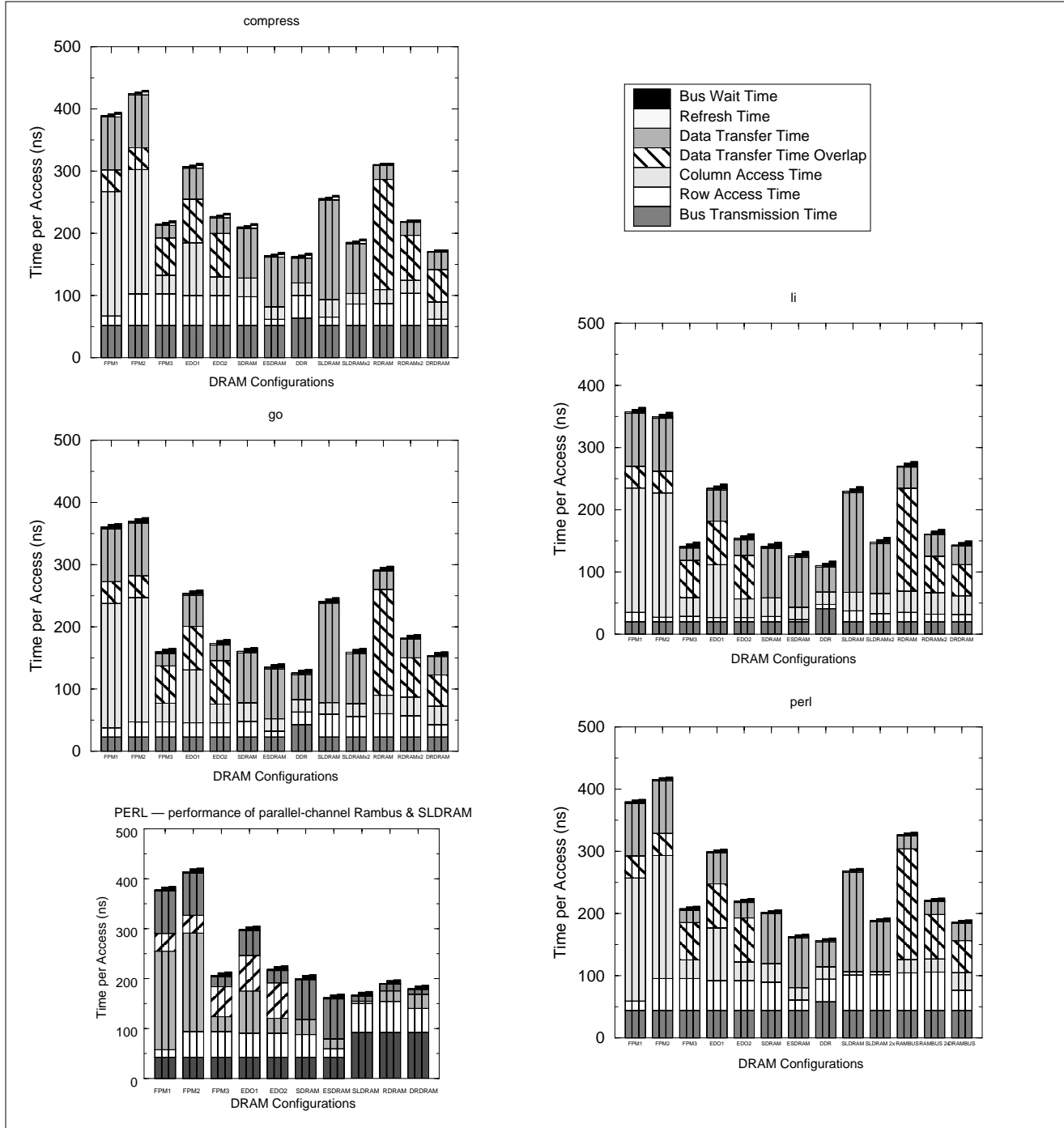


Figure 11: Break-downs for primary memory access time, 128-BIT bus. These graphs present the average access time on a 128-bit bus across DRAM architectures for benchmarks that display the most widely varying behavior. The different DRAM architectures display significantly different access times. The 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.

variations in performance are not large, we only show 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 that *perl* and *compress* have heavier write traffic than *go* or *li*.

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(d). These “parallel-channel” results are intended to demonstrate the mismatch between today’s bus speeds and fastest DRAMs; they are shown in the bottom left corner 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 DDR results *Bus Transmission* accounts for 40–45% of the total, and 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. 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 [16].

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 data 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 “x2” variants of SLDRAM and RDRAM demonstrate excellent performance as well. Both *Column Access* and *Data Transfer* decrease by a factor of two; both channels can be active simultaneously, fetching or writing different parts of the same L2 cache line. This behavior is expected. This reduces the average DRAM access time by roughly 30% and the total execution time (see Figure 10) by 25%, making these configurations as fast as any other of the modern DRAM designs.

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 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. DDR is the fastest of the DRAM architectures studied, which is not surprising due to its bandwidth, which is twice that of the other DRAMs studied. It is interesting to note that its performance is slightly better than that of Enhanced Memory’s SDRAM, and Figure 10 shows that while it has reduced the bandwidth portion of latency more than ESDRAM, ESDRAM has reduced the latency component more than DDR. This is to be expected, as DDR has a core that is fundamentally similar to that of SDRAM—it simply has a faster interface—while ESDRAM has a core unlike any other DRAM architecture studied: latching the entire row optimally hides the precharge activity and increases the overlap between access to different rows, thus reducing average latency.

As modeled, SLDRAM and Rambus designs have higher end-to-end transaction latencies than SDRAM, ESDRAM, or DDR, as they require twice as many data transfers 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. The SLDRAMx2 and RDRAMx2 variants, which have the same bandwidth and therefore the same number of

data transfers as the other organizations, manage to make up the difference in performance, with SLDRAMx2 yielding the same performance as Direct Rambus. Direct Rambus also comes out about equal to SDRAM in end-to-end latency and a little behind ESDRAM and DDR.

Last, the DDR results and parallel-channel results demonstrate the failure of a 100MHz 128-bit bus to keep up with today's fastest parts. DDR spends more than 40% of its time in bus transmission—sometimes as much as twice the overhead as other DRAMs, suggesting that the bus is not keeping up with the speed of the DDR DRAM core. In the parallel-channel organizations, 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. Both these results suggest that we are pushing the limits of today's buses. The Alpha 21364 will solve this problem by ganging together multiple Rambus Channels connected directly to the CPU, eliminating the 100MHz bus [16].

5.4 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 but not all of the 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 is no mistake. Two configurations are interleaved; the others are not. Making the bus the width of the request size obviates interleaving.

The fastest of the designs is ESDRAM, not DDR as one would expect based on the average access time graphs. As mentioned earlier, this is because ESDRAM is the one architecture studied that has a different internal core; all other DRAMs have the same DRAM core inside. DDR therefore only has a bandwidth advantage over others—an advantage that is nullified when modeling a perfect-width bus. This figure thus serves to highlight the time-to-first-bit inefficiencies of the various DRAM interfaces.

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

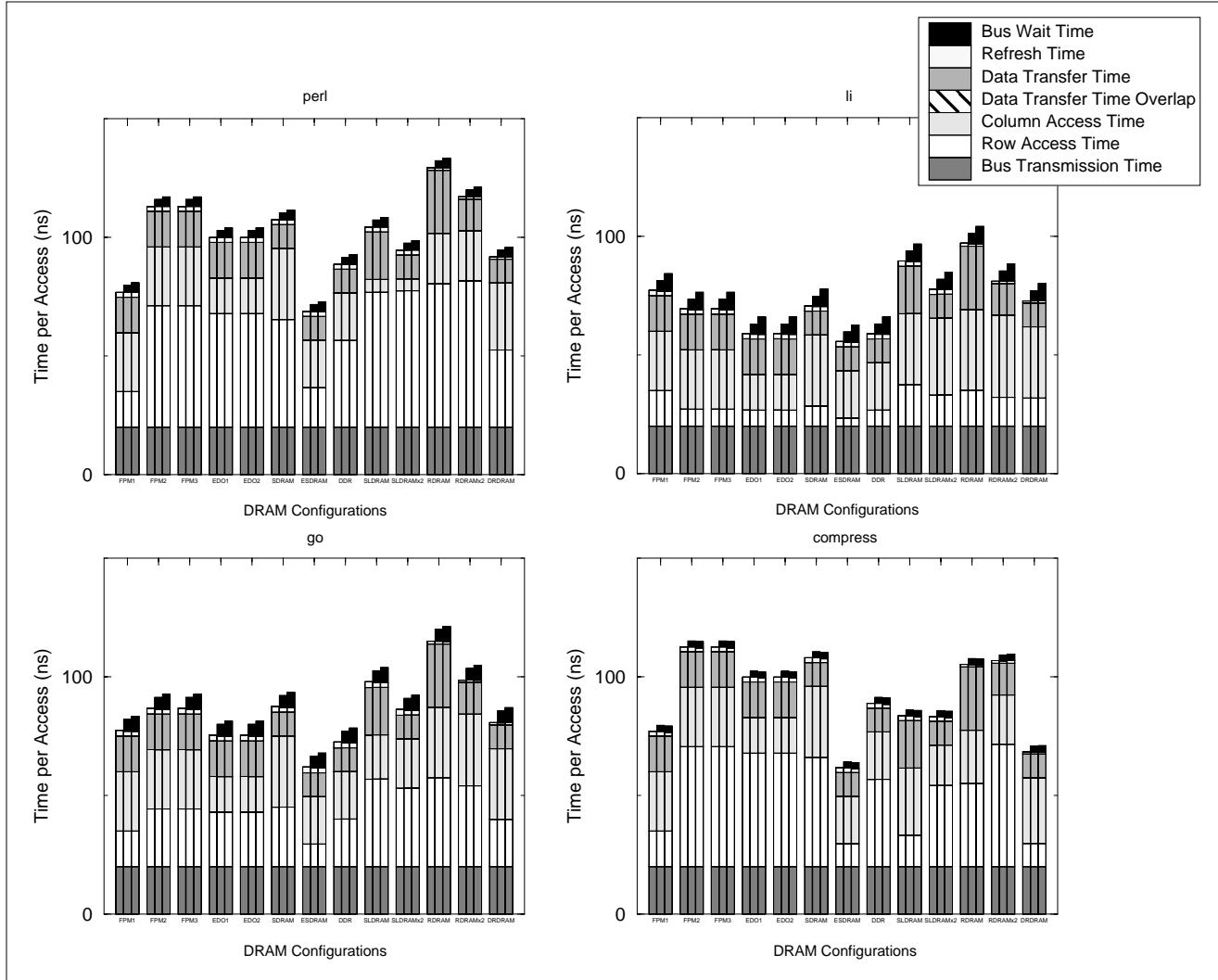


Figure 12: Break-downs for primary memory access time, 128-BYTE bus. These graphs present the average access time on a 128-byte bus, the same width as an L2 cache line. Therefore the pipelined access to memory (multiple column accesses per row access) is not seen, and the *Row Access* component becomes 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.

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 19) can have a dramatic impact on the access latency—for example, there is a 10% to 90% difference between the average access latencies for *li* 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 [24, 25].

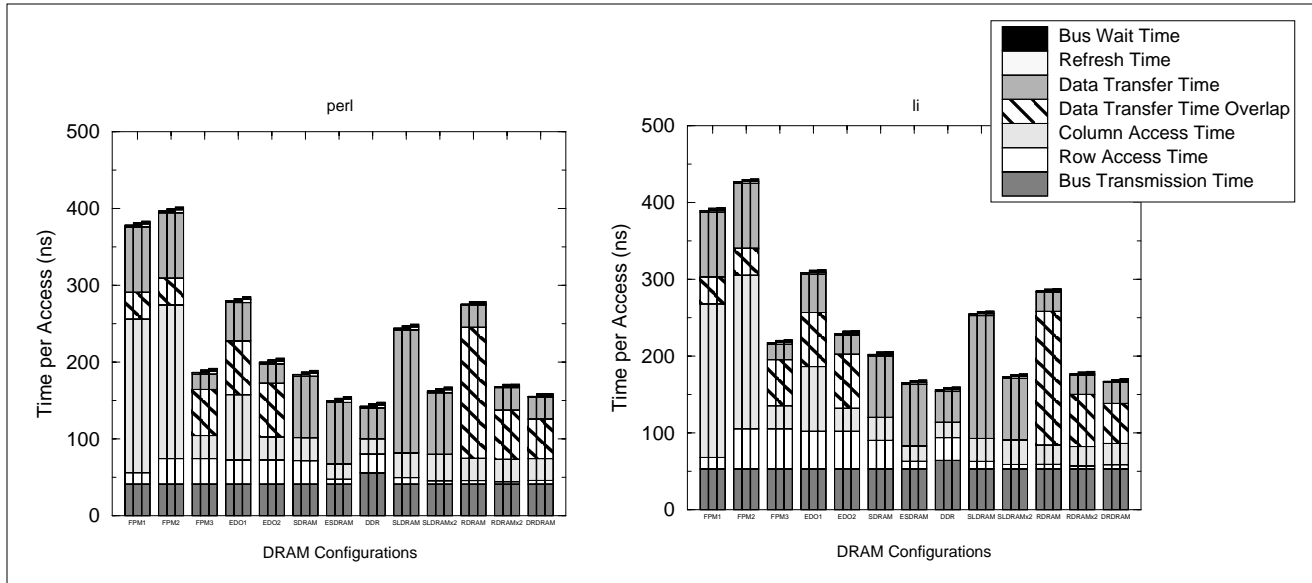


Figure 13: Break-downs for primary memory access time, 4MB L2 caches. These graphs present the average access time on a 128-bit bus across DRAMs.

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 (see Figure 13); however, this simply serves to illustrate the behavior when the bulk of the requests reaching the DRAM system are compulsory cache misses.

Comparing the 128-byte 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

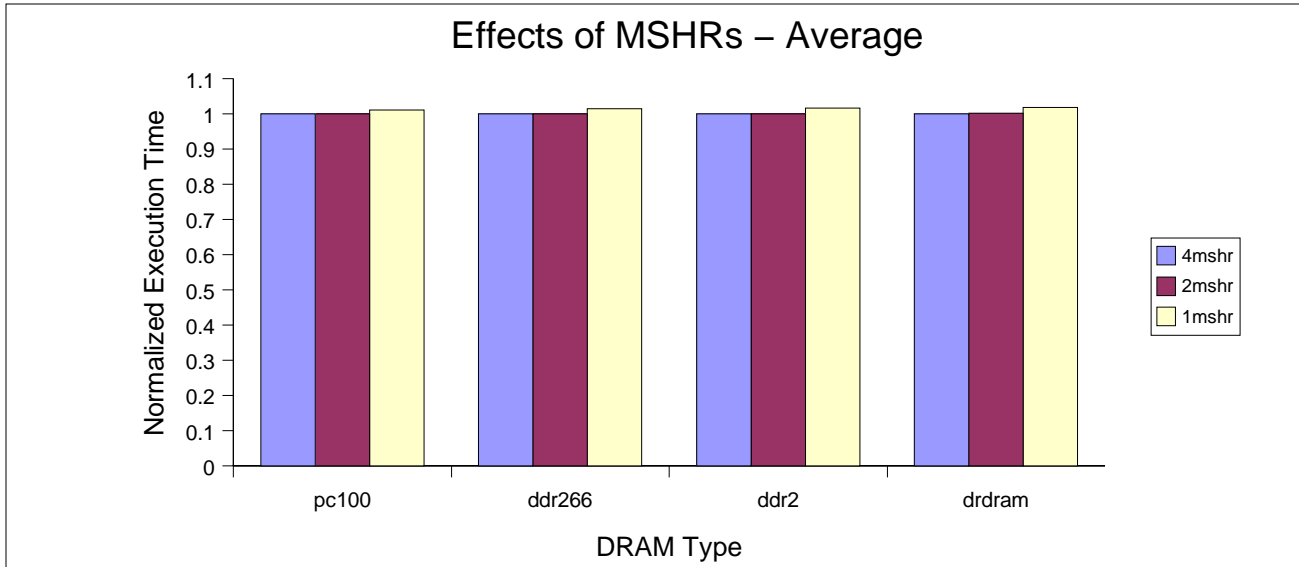


Figure 14: The effect of limiting MSHRs to 1. The graph shows the effect on performance of varying the number of MSHRs at the L2 cache from one to four. The graphs represent average results of several benchmarks, shown in the following figure.

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—for example, our recent comparison of VDRAM and ESDRAM shows little difference in performance, though VDRAM expends more die area in a set-associative organization of the same number of SRAM bits on the DRAM die [11].

5.5 The Effect of Limited MSHRs

As mentioned in section 4.1, the measurements presented so far represent a system model with lock-up free caches, but with what is effectively a single MSHR at the L2-cache level. Though there can be up to 32 outstanding misses between the L1 and L2 caches, and though the L2 cache allows any number of hits under a miss, only a single L2 miss can be active in the memory system. This fails to exploit the degrees of concurrency offered by high performance DRAM architectures—for example, Direct Rambus can support up to three concurrent accesses.

It is reasonable to wonder how badly this limitation hampers the performance of the DRAMs under study. To quantify the effect, we present additional data (using a different simulator) for the newer DRAM architectures in a highly concurrent environment in which we vary the number of MSHRs. Most of this data, as well as the particulars of the simulator environment, can be found in [40]. The results in Figure 14 present data for PC100, Direct Rambus DRAM, DDR266, and DDR2 averaged over a number of benchmarks. The results in Figure 15 show the individual benchmarks for DRDRAM alone. Obviously, we

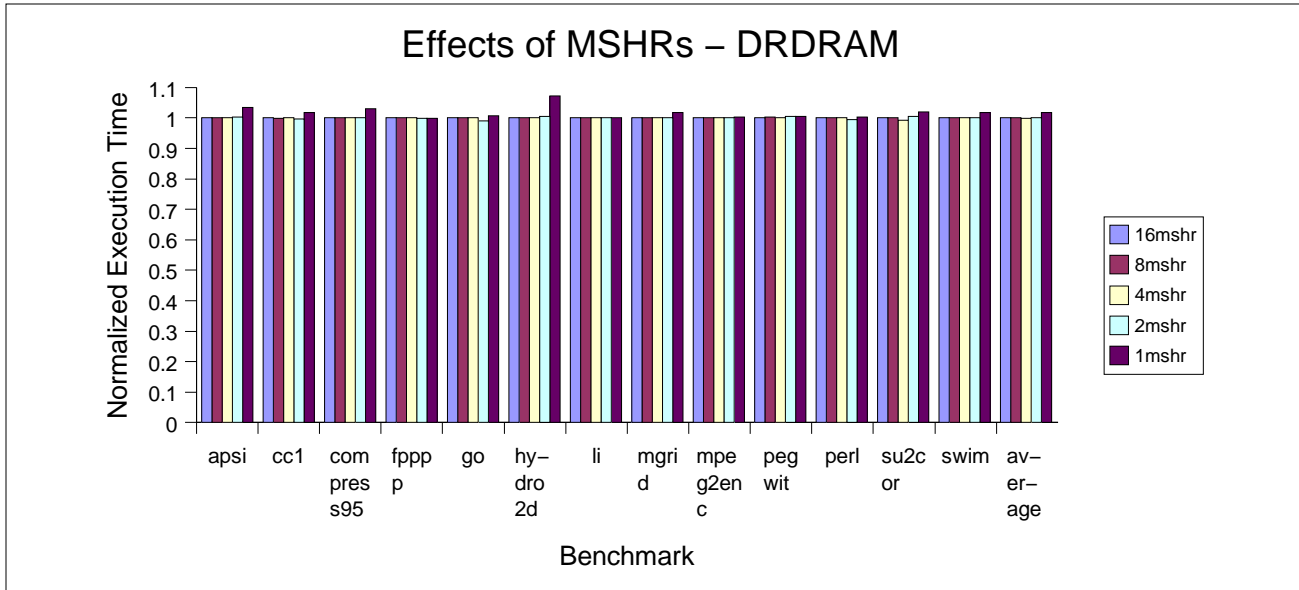


Figure 15: The effect of MSHRs on Direct Rambus. The graph shows the performance effect of varying MSHRs in a Direct Rambus-based system. The results are for each individual benchmark, as well as the average.

expect little variation from four to sixteen MSHRs because this exceeds the capabilities of single-bus designs—nonetheless, it acts as a reasonable sanity-check.

As the graphs show, there is on average a 1% difference in execution time between a system with a single MSHR and a system with enough MSHRs to fully occupy a DRAM architecture’s abilities. We measured a maximum difference of roughly 5% (shown in the DRDRAM results). We conclude that our MSHR-based limitation of concurrency in the DRAM system introduces no significant performance degradation. This is not to say that concurrency in the memory system is not beneficial, however: We look more closely at the effects of memory-system concurrency in several follow-on studies that suggest concurrency is better exploited at the DRAM-system level than the DRAM-architecture level [7, 8].

5.6 Critical-Word Latencies

The average access time numbers shown in Figure 11 represent average end-to-end latency: e.g., for a read they represent the time from the start of the DRAM request to the moment the last word in the requested block reaches the level-2 cache. This is somewhat misleading because it is widely held that the true limiter to performance is the critical-word latency.

Critical-word latencies are shown in Figure 16 for most of the DRAM architectures, at the highest CPU speed. The figure shows that time-to-critical-word is significantly lower than the end-to-end latency, as expected. At great expense, the end-to-end latency can be improved by widening the bus, thereby making

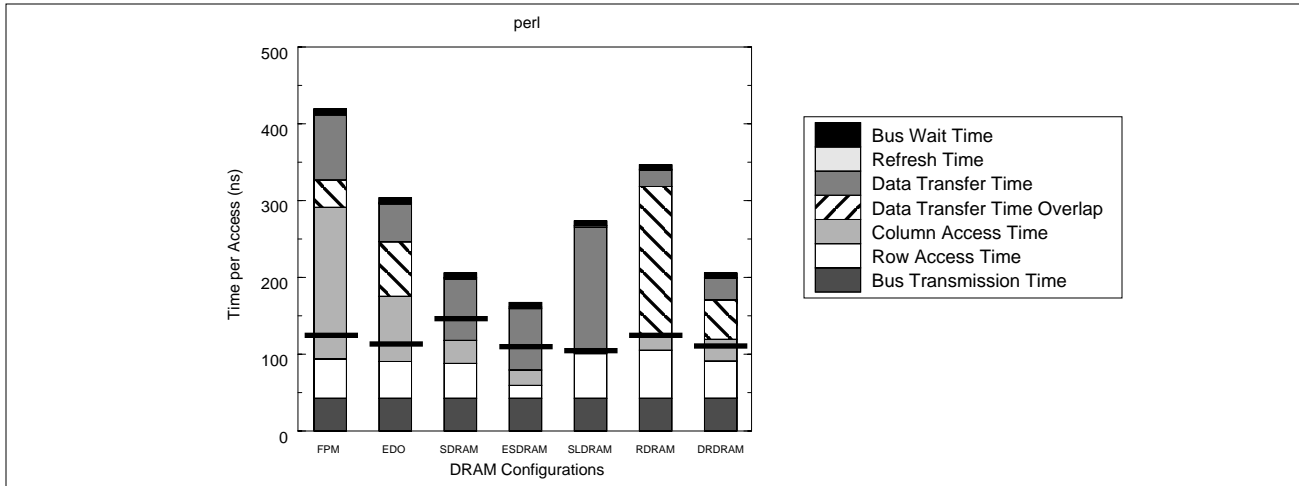


Figure 16: Critical-word latencies. On top of the average end-to-end latencies that were shown in Figure 11, we have drawn solid black bars representing the time at which the critical word arrived at the CPU.

the end-to-end latency equal to the critical-word latency. This is shown in Figure 12 (described earlier). Note that doing so yields latencies similar to the critical-word latencies in Figure 16—in short, there is no significant latency argument for widening the bus. To reduce latency, one must speed up the bus, speed up the DRAM core, improve the hit ratio in the DRAM row buffers, or redesign the DRAM interface.

It is interesting to note that the SLDRAM and Rambus designs excel 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. However, as one can see by looking at Figures 16 and 10 side-by-side, the total execution time seems to correlate more with the end-to-end latency than the critical-word latency—e.g., if total execution time scaled with critical-word latency, we would expect SLDRAM results to be faster than ESDRAM (which they are not), and we would expect SDRAM results to be 10-20% slower than ESDRAM, SLDRAM, RDRAM, and DRDRAM (which they are not). We would expect the ranking from fastest system to slowest to be

1. SLDRAM
2. DRDRAM tied with ESDRAM
3. RDRAM
4. SDRAM

when, in fact, the order (for both PERL and GCC, at both medium and high CPU speeds) is

1. (DDR, not shown in the time-to-critical-word figure)

2. ESDRAM
3. DRDRAM tied with SDRAM
4. SLDRAM
5. RDRAM

The fact that, in these cases, the total execution time correlates better with end-to-end latency than with critical-word latency simply suggests that, on average, these benchmarks tend to use a significant portion of each L2 cache line.

5.7 Cost-Performance Considerations

The organizations are equal in their capacity: all but DDR and the interleaved examples use eight 64Mbit DRAMs. The FPM3 organization uses 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 organizations 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, ESDRAM, and DDR 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 DRAM device, the FPM, EDO, SDRAM, ESDRAM, and DDR 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 19); 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, Rambus parts typically carry a stiff price premium—roughly 3x at the time of this writing, despite less than a 20% area premium—but significantly less than the 8x disparity in the number of chips required to achieve the same performance.

5.8 Using the Collective Row Buffers in Lieu of an L2 Cache

Associated with each DRAM core is a set of sense amps that can latch data; this amounts to a cache of high-speed memory, and internally-banked DRAMs have several of these caches. Moreover, the trend in the most recent DRAMs is to add even more on-chip storage (in addition to the sense amps) via SRAM structures in various organizations. Collectively, a DRAM or bank of DRAMs can have a sizable cache in these sense amps and SRAM buffers. For each DRAM architecture studied, the amount of row-buffer storage 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). ESDRAM adds an SRAM buffer to the sense amps and so effectively doubles the storage. Newer designs, such as VCDRAM, place even more SRAM on-chip [12].

Many computer architects and DRAM manufacturers have suggested that the new DRAMs, with their extensive use of row buffering, enable lower-cost systems that forgo L2 caches but nonetheless have high performance [15, 18, 29]. The argument is that a system with numerous DRAM chips, each with its large number of open bits stored in row buffers and SRAM caches, effectively already has a level-2 cache. The size of this cache, on a memory-module basis, is equal to the size of each DRAM’s internal row-buffer storage (both SRAM-based and sense-amp-based, depending on organization) times the number of DRAMs on each memory module (e.g. DIMM). The total cache size in a system is thus the size of each “DIMM-cache” times the number of memory modules in the system. The benefit offered over a traditional L2 organization is that the size of the cache inherently scales with the size of the system’s memory. The next experiment revisits the DRAM systems already presented; for each DRAM it gives the hit rates of the DIMM-caches and shows the performance result of removing the L2 cache from the system.

Figure 17 shows the total execution time of each benchmark for an SDRAM organization. Clearly, for 10GHz CPUs, today’s DRAMs will not keep up without a level-2 cache. However, for the 100MHz and 1GHz CPU speeds (the left and middle bars in each group of three), we see that the memory component is not overwhelming. With a 1GHz CPU and no L2 cache, the DRAM system accounts for 10-80% of the total execution time. For low-cost systems, this might very well be acceptable.

Figure 18 shows the average access times for both 128-bit buses and ideal 64-byte buses. The ideal buses are 64 bytes and not 128 bytes because the L1 cache block is 64 bytes. The main result is that these graphs look just like previous graphs, except that the scale is smaller because of the difference in L1 and L2 block sizes (with the L2 gone, the amount of data per request is cut in half). The most obvious

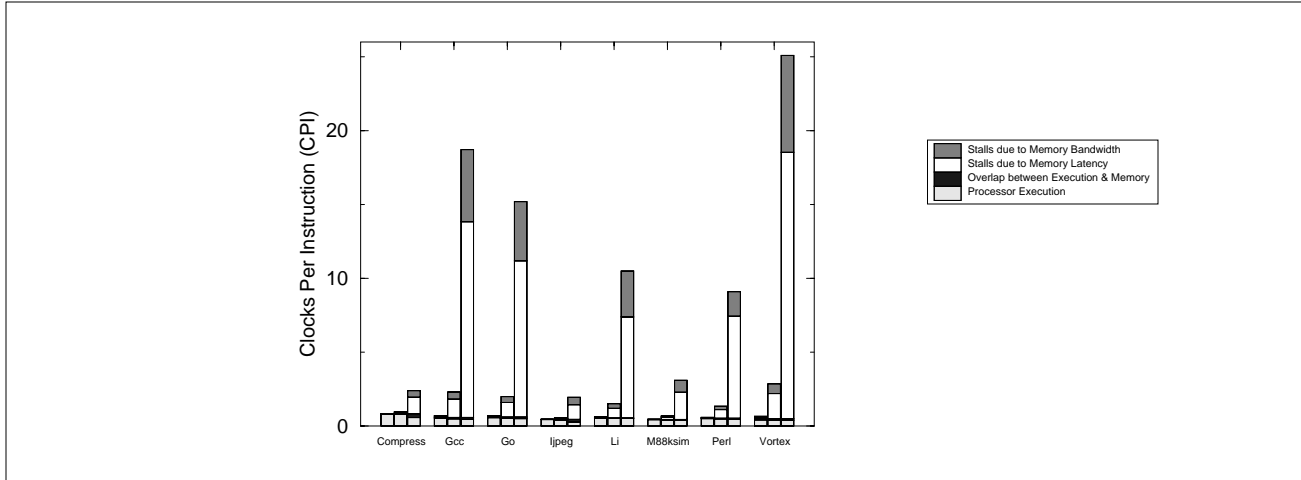


Figure 17: Performance without a level-2 cache.

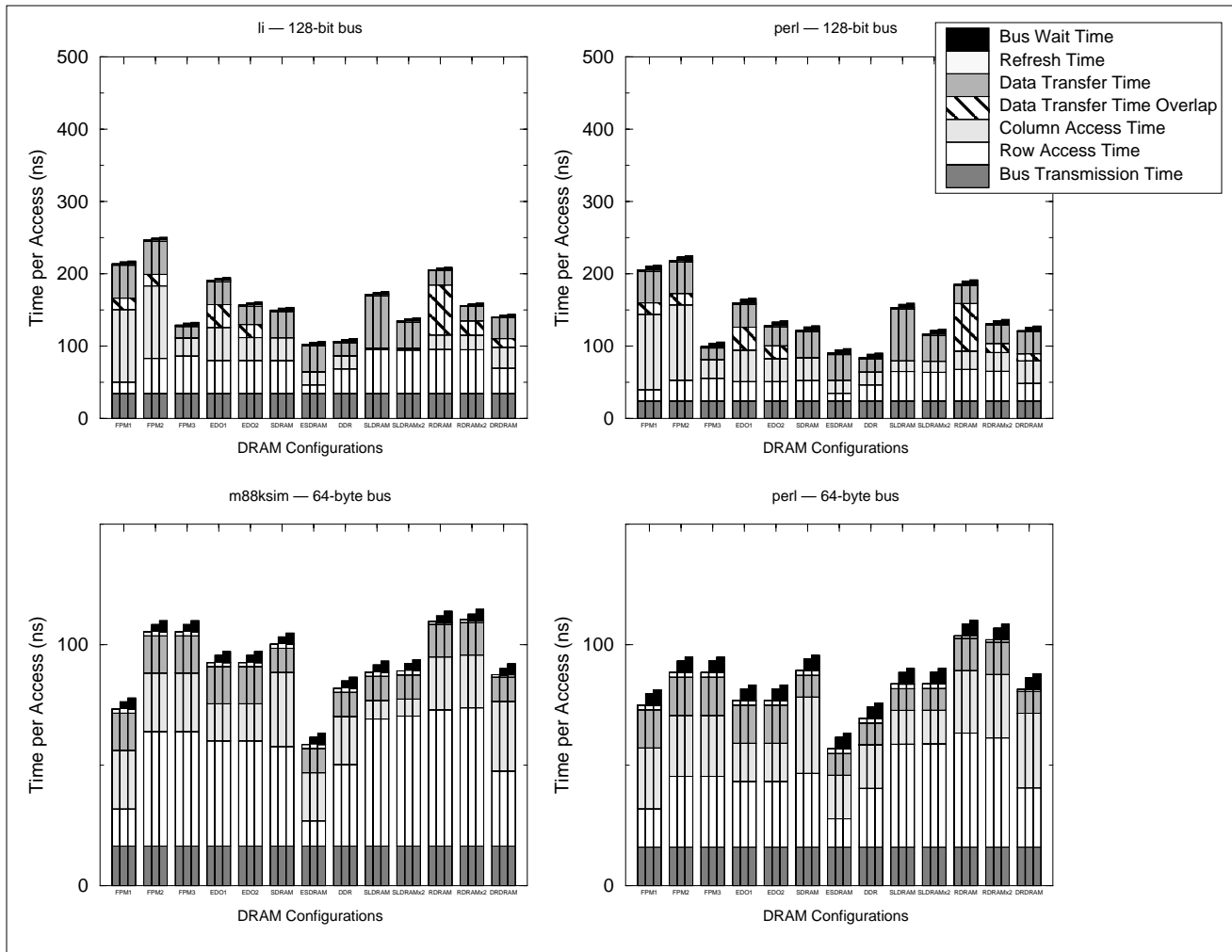


Figure 18: Break-downs for primary memory access time, 128-BIT and 64-BYTE bus. These graphs present the average access time on a 128-bit bus across DRAM architectures for the three benchmarks that display the most widely varying behavior. The different DRAM architectures display significantly different 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.

difference between these results and previous results is that there is very little variation from benchmark to benchmark. This is largely because the elimination of the L2 cache makes write operations more frequent, thereby disrupting read locality [9]. This is also seen in the decreased hit rates relative to hit rates with 1MB and 4MB L2 caches (next figure).

Figure 19 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 ideal buses. We present results for two sets of benchmarks, including applications from SPEC and Etch suites. As mentioned later, the Etch applications are included because they tend to have larger footprints than SPEC.

The results show that memory requests frequently hit the row buffers; hit rates range from 2–97%, with a mean of 40%. Hit rates increase with increasing L2 cache size (because the DRAM traffic is increasingly compulsory misses, which tend to be sequential) and decrease as the L2 cache disappears (because the writeback L2 does a good job of filtering out writes, as well as the fact that more non-compulsory misses will hit the DRAM with the L2 cache gone). As shown in our previous study [9], 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.

5.9 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 [14] 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-buffer hit rates had a

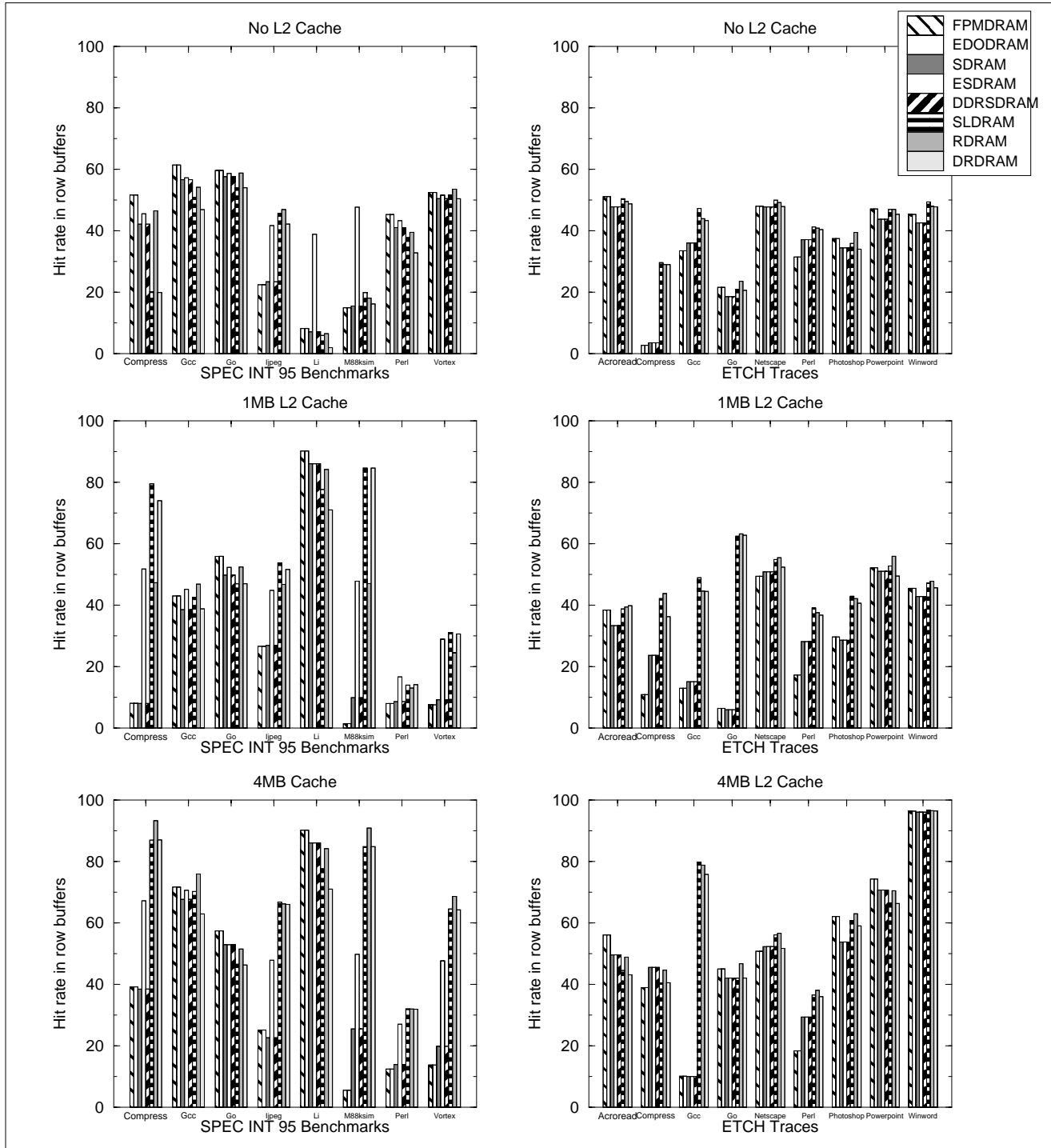


Figure 19: Hit-rates in the row buffers. These graphs show hit-rates for the benchmarks on each of the DRAM architectures. The newer DRAMs, with more internal banking, tend to have higher hit rates. Write traffic, due to writebacks, disrupts the locality of the address stream for architectures with fewer internal banks.

smaller standard deviation. An example for the *compress* benchmark is shown in Figure 20; this graph is very representative of the entire Etch suite. In general, the Etch benchmarks have similar break-downs,

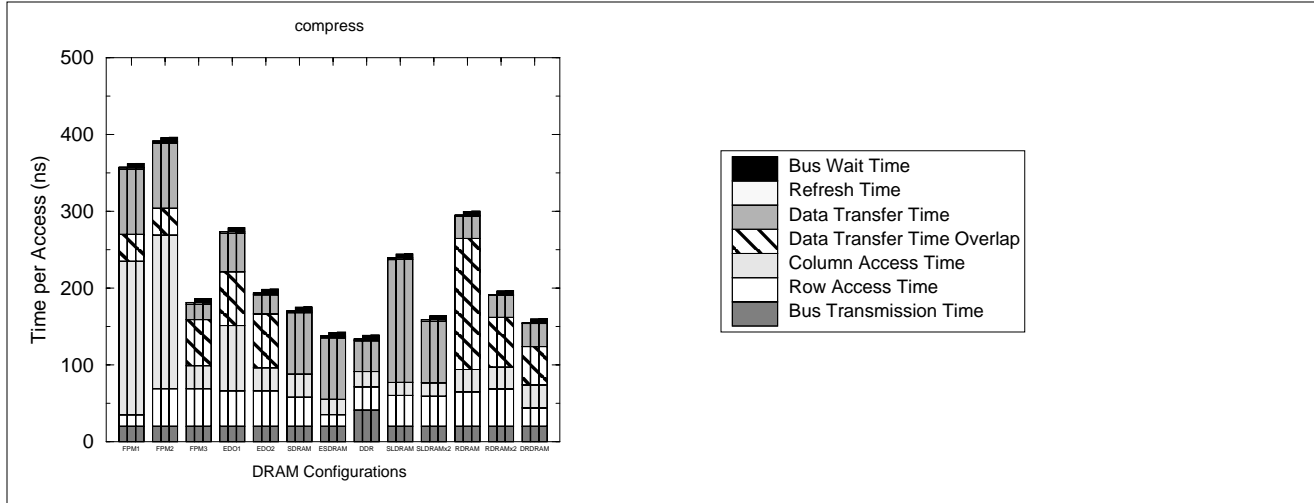


Figure 20: Average access times for ETCH traces. Results are for 1MB L2 caches and 128-bit buses.

which is expected since their row-buffer hit rates have a small standard deviation. Also, the average access times for the Etch benchmarks tend to be smaller than their SimpleScalar counterparts (see Figure 11), and the differences lie primarily in the *Bus Transmission Time* component. Trace-driven simulations are often derided for being less accurate; the fact that these results are so similar to those obtained through accurate request timing in an out-of-order core suggests that trace-driven approaches may be viable for future DRAM studies. This is corroborated by other results of ours [11].

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.

We have also extended the work to incorporate higher degrees of concurrency on the memory bus and additional experimental DRAM architectures [7, 8, 11].

As buses grow wider, *Row Access Time* becomes significant; in our 1MB L2 studies it accounts for 20–50% of the total latency. 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 21. 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

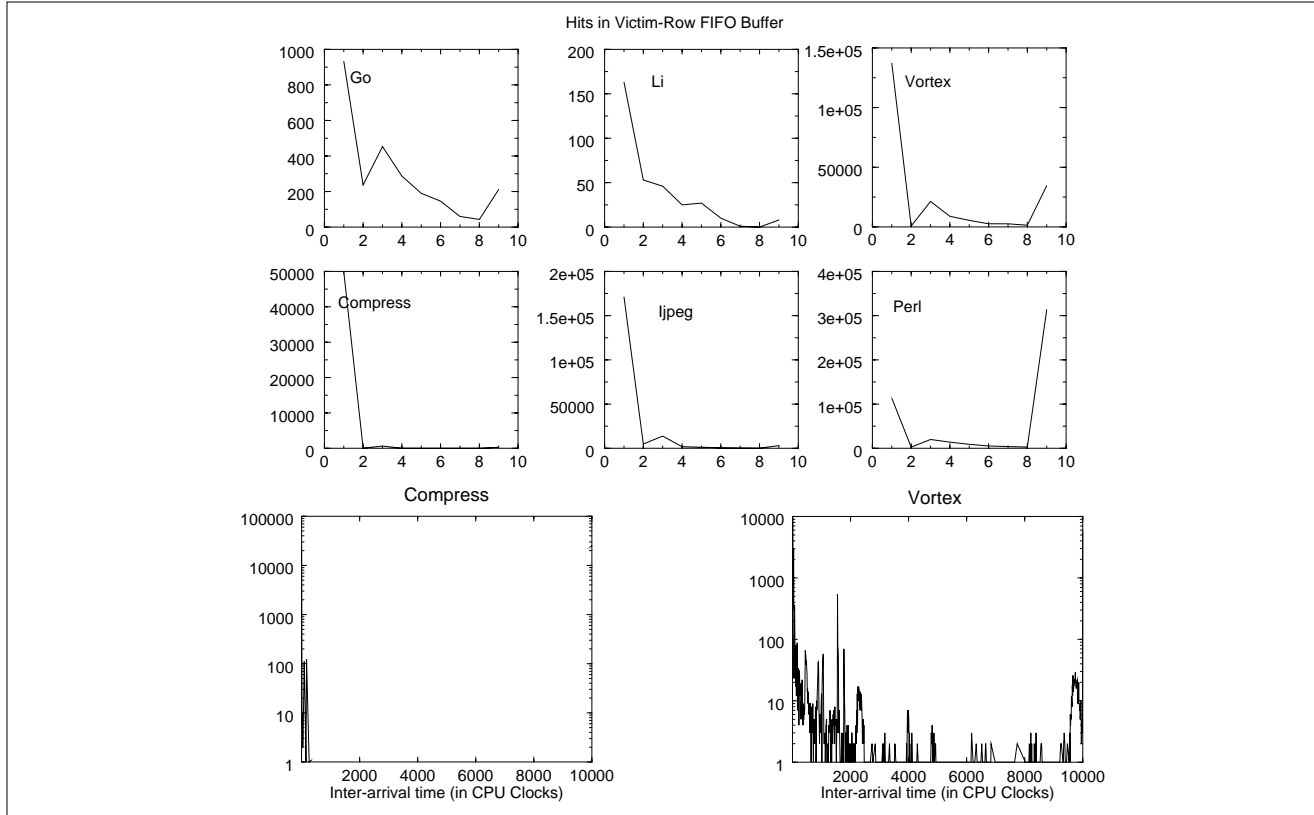


Figure 21: Locality in the stream of accesses to the single open row in the FPM DRAM. The top six graphs show the frequency with which accesses to a given DRAM page hit at stack depth x . The bottom two graphs show the inter-arrival time of accesses that hit in an open DRAM page. Both sets of graphs show that when references are made to the data in a particular DRAM page, the accesses tend to be localized in time.

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 earlier version of this paper that appeared in *ISCA '99* [9].

Trevor Mudge is supported in part by DARPA grant DABT 63-96-C0047. Vinodh Cuppu and Bruce Jacob are supported in part by NSF CAREER Award CCR-9983618 and NSF grant EIA-9806645.

REFERENCES

- [1] L. A. Barroso, K. Gharachorloo, and E. Bugnion. "Memory system characterization of commercial workloads." In *Proc. 25th Annual International Symposium on Computer Architecture (ISCA '98)*, Barcelona, Spain, June 1998, pp. 3–14.
- [2] D. Bhandarkar and J. Ding. "Performance characterization of the Pentium Pro processor." In *Proc. Third International Symposium on High Performance Computer Architecture (HPCA '97)*, San Antonio TX, February 1997, pp. 288–297.
- [3] N. Bowman, N. Cardwell, C. Kozyrakis, C. Romer, and H. Wang. "Evaluation of existing architectures in IRAM systems." In *Workshop on Mixing Logic and DRAM*, Denver CO, 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, J. R. Goodman, and A. Kagi. "Memory bandwidth limitations of future microprocessors." In *Proc. 23rd Annual International Symposium on Computer Architecture (ISCA '96)*, Philadelphia PA, May 1996, pp. 78–89.
- [6] R. Crisp. "Direct Rambus technology: The new main memory standard." *IEEE Micro*, vol. 17, no. 6, pp. 18–28, November 1997.
- [7] V. Cuppu and B. Jacob. "Concurrency, latency, or system overhead: Which has the largest impact on uniprocessor DRAM-system performance?" In *Proc. 28th International Symposium on Computer Architecture (ISCA '01)*. Goteborg Sweden, June 2001.
- [8] V. Cuppu and B. Jacob. "Organizational design trade-offs at the DRAM, memory bus, and memory controller level: Initial results." Tech. Rep. UMD-SCA-1999-2, University of Maryland Systems & Computer Architecture Group, November 1999.
- [9] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. "A performance comparison of contemporary DRAM architectures." In *Proc. 26th Annual International Symposium on Computer Architecture (ISCA '99)*, Atlanta GA, May 1999, pp. 222–233.
- [10] Z. Cvetanovic and D. Bhandarkar. "Performance characterization of the Alpha 21164 microprocessor using TP and SPEC workloads." In *Proc. Second International Symposium on High Performance Computer Architecture (HPCA '96)*, San Jose CA, February 1996, pp. 270–280.
- [11] B. Davis, T. Mudge, B. Jacob, and V. Cuppu. "DDR2 and low latency variants." In *Proc. Memory Wall Workshop at the 26th Annual Int'l Symposium on Computer Architecture*, Vancouver, Canada, May 2000.
- [12] B. Dipert. "The slammin, jammin, DRAM scramble." *EDN*, vol. 2000, no. 2, pp. 68–82, January 2000.
- [13] ESDRAM. *Enhanced SDRAM 1M x 16*. Enhanced Memory Systems, Inc., http://www.edram.com/products/datasheets/16M_esdram0298a.pdf, 1998.
- [14] Etch. *Memory System Research at the University of Washington*. The University of Washington, <http://etch.cs.washington.edu/>, 1998.
- [15] J. R. Goodman and M. Chiang. "The use of static column RAM as a memory hierarchy." In *Proc. 11th Annual International Symposium on Computer Architecture (ISCA '84)*, Ann Arbor MI, June 1984, pp. 167–174.
- [16] L. Gwennap. "Alpha 21364 to ease memory bottleneck: Compaq will add Direct RDRAM to 21264 core for late 2000 shipments." *Microprocessor Report*, vol. 12, no. 14, pp. 12–15, October 1998.
- [17] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, 2nd Edition*. Morgan Kaufmann Publishers, Inc., 1996.
- [18] W.-C. Hsu and J. E. Smith. "Performance of cached DRAM organizations in vector supercomputers." In *Proc. 20th*

Annual International Symposium on Computer Architecture (ISCA'93), San Diego CA, May 1993, pp. 327–336.

- [19] IBM. *EDO DRAM 4M x 16 Part No. IBM0165165PT3C*. International Business Machines, <http://www.chips.ibm.com/products/memory/88H2011/88H2011.pdf>, 1998.
- [20] IBM. *SDRAM 1M x 16 x 4 Bank Part No. IBM0364164*. International Business Machines, <http://www.chips.ibm.com/products/memory/19L3265/19L3265.pdf>, 1998.
- [21] IBM. *DDR DRAM 16M x 8 Part No. IBM0612804GT3B*. International Business Machines, <http://www.chips.ibm.com/products/memory/06K0566/06K0566.pdf>, 2000.
- [22] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker. “Performance characterization of a quad Pentium Pro SMP using OLTP workloads.” In *Proc. 25th Annual International Symposium on Computer Architecture (ISCA'98)*, Barcelona, Spain, June 1998, pp. 15–26.
- [23] C. Kozyrakis, S. Perissakis, D. Patterson, T. Anderson, K. Asanovic, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, R. Thomas, N. Treuhhaft, and K. Yelick. “Scalable processors in the billion-transistor era: IRAM.” *IEEE Computer*, vol. 30, no. 9, pp. 75–78, September 1997.
- [24] S. McKee, A. Aluwihare, B. Clark, R. Klenke, T. Landon, C. Oliver, M. Salinas, A. Szymkowiak, K. Wright, W. Wulf, and J. Aylor. “Design and evaluation of dynamic access ordering hardware.” In *Proc. International Conference on Supercomputing*, Philadelphia PA, May 1996.
- [25] S. A. McKee and W. A. Wulf. “Access ordering and memory-conscious cache utilization.” In *Proc. International Symposium on High Performance Computer Architecture (HPCA'95)*, Raleigh NC, January 1995, pp. 253–262.
- [26] B. Nayfeh, L. Hammond, and K. Olukotun. “Evaluation of design alternatives for a multiprocessor microprocessor.” In *Proc. 23rd Annual International Symposium on Computer Architecture (ISCA'96)*, Philadelphia PA, May 1996, pp. 67–77.
- [27] B. A. Nayfeh, K. Olukotun, and J. P. Singh. “The impact of shared-cache clustering in small-scale shared-memory multiprocessors.” In *Proc. Second International Symposium on High Performance Computer Architecture (HP-CA'96)*, San Jose CA, February 1996, pp. 74–84.
- [28] S. Przybylski. *New DRAM Technologies: A Comprehensive Analysis of the New Architectures*. MicroDesign Resources, Sebastopol CA, 1996.
- [29] Rambus. “Rambus memory: Enabling technology for PC graphics.” Tech. Rep., Rambus Inc., Mountain View CA, October 1994.
- [30] Rambus. “Comparing RDRAM and SGRAM for 3D applications.” Tech. Rep., Rambus Inc., Mountain View CA, October 1996.
- [31] Rambus. “Memory latency comparison.” Tech. Rep., Rambus Inc., Mountain View CA, September 1996.
- [32] Rambus. *16/18Mbit & 64/72Mbit Concurrent RDRAM Data Sheet*. Rambus, <http://www.rambus.com/docs/Cnctds.pdf>, 1998.
- [33] Rambus. *Direct RDRAM 64/72-Mbit Data Sheet*. Rambus, <http://www.rambus.com/docs/64dDDS.pdf>, 1998.
- [34] P. Ranganathan, K. Gharachorloo, S. V. Adve, and L. A. Barroso. “Performance of database workloads on shared-memory systems with out-of-order processors.” In *Proc. Eighth Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'98)*, San Jose CA, October 1998, pp. 307–318.
- [35] M. Rosenblum, E. Bugnion, S. A. Herrod, E. Witchel, and A. Gupta. “The impact of architectural trends on operating system performance.” In *Proc. 15th ACM Symposium on Operating Systems Principles (SOSP'95)*, December 1995.
- [36] Samsung. *FPM DRAM 4M x 16 Part No. KM416V4100C*. Samsung Semiconductor, [http://www.usa.samsung-semi.com/products/prodspec/dramcomp/KM416V40\(1\)00C.PDF](http://www.usa.samsung-semi.com/products/prodspec/dramcomp/KM416V40(1)00C.PDF), 1998.
- [37] A. Saulsbury, F. Pong, and A. Nowatzky. “Missing the memory wall: The case for processor/memory integration.” In *Proc. 23rd Annual International Symposium on Computer Architecture (ISCA'96)*, Philadelphia PA, May 1996, pp. 90–101.
- [38] SLDRAM. *4M x 18 SLDRAM Advance Datasheet*. SLDRAM, Inc., <http://www.sldram.com/Docu->

ments/corp400b.pdf, 1998.

- [39] R. Wilson. "MoSys tries synthetic SRAM." EE Times Online, July 15, 1997, July 1997. <http://www.eetimes.com/news/98/1017news/tries.html>.
- [40] B. Davis. *Modern DRAM Architectures*. Ph.D. Thesis, University of Michigan. 2001.

Author Information

Vinodh Cuppu received the M.S. degree in Electrical Engineering from the University of Maryland, College Park in 2000, and the B.E. degree in Electronics and Communication Engineering from the University of Madras, India in 1997. He is currently a Ph.D. student in the Department of Electrical and Computer Engineering at the University of Maryland, College Park. His research interests include DRAM architectures, latency reduction techniques and power reduction techniques for embedded microprocessors. He is a member of the IEEE and the ACM.

Bruce Jacob received the A.B. degree *cum laude* in Mathematics from Harvard University in 1988, and the M.S. and Ph.D. degrees in Computer Science and Engineering from the University of Michigan, Ann Arbor, in 1995 and 1997, respectively. At the University of Michigan he was part of a design team building high-performance, high-clock-rate microprocessors. He has also worked as a software engineer for two successful startup companies: Boston Technology and Priority Call Management. At Boston Technology he worked as a distributed systems developer, and at Priority Call Management he was the initial system architect and chief engineer. He is currently on the faculty of the University of Maryland, College Park, where he is an Assistant Professor of Electrical and Computer Engineering. His present research covers memory-system design and includes DRAM architectures, virtual memory systems, and memory management hardware and software for real-time and embedded systems. He is a recipient of an NSF CAREER award for his work on DRAM systems. He is a member of the IEEE, the ACM, and Sigma Xi.

Brian Davis received the B.S.E. degree *magna cum laude* in Electrical Engineering from Michigan Technological University in 1991, and the M.S.E. and Ph.D. degrees in Computer Engineering from University of Michigan, Ann Arbor, in 1994 and 2001 respectively. He is presently an Assistant Professor in the Electrical and Computer Engineering department at Michigan Technological University. He has served on the faculty of University of Toledo and Willamette University. His present research is focussed upon modern DRAM interfaces and architectures, latency tolerance in high-speed microprocessors, memory controller policies, and computer simulation methodologies. Brian Davis is a member of the IEEE, the ACM and Tau Beta Pi.

Trevor Mudge received the B.Sc. degree in Cybernetics from the University of Reading, England, in 1969, and the M.S. and Ph.D. degrees in Computer Science from the University of Illinois, Urbana, in 1973 and 1977, respectively. Since 1977, he has been on the faculty of the University of Michigan, Ann Arbor. He is presently a Professor of Electrical Engineering and Computer Science and the Director of the Advanced Computer Architecture Laboratory — a group of eight faculty and 70 graduate research assistants. He is author of more than 150 papers on computer architecture, programming languages, VLSI design, and computer vision, and he holds a patent in computer-aided design of VLSI circuits. He has also chaired about 20 theses in these research areas. His research interests include computer architecture, computer-aided design, and compilers. In addition to his position as a faculty member, he is a consultant for several computer companies. He is also Associate Editor for *IEEE Transaction on Computers* and *ACM Computing Surveys*. Trevor Mudge is a Fellow of the IEEE, a member of the ACM, the IEE, and the British Computer Society.