A Case for Studying DRAM Issues at the System Level

THE WIDENING GAP BETWEEN TODAY'S PROCESSOR AND MEMORY SPEEDS MAKES DRAM SUBSYSTEM DESIGN AN INCREASINGLY IMPORTANT PART OF COMPUTER SYSTEM DESIGN. IF THE DRAM RESEARCH COMMUNITY WOULD FOLLOW THE MICROPROCESSOR COMMUNITY'S LEAD BY LEANING MORE HEAVILY ON ARCHITECTURE- AND SYSTEM-LEVEL SOLUTIONS IN ADDITION TO TECHNOLOGY-LEVEL SOLUTIONS TO ACHIEVE HIGHER PERFORMANCE, THE GAP MIGHT BEGIN TO CLOSE.

Bruce Jacob
University of Maryland

In 1996, Richard Sites, one of the fathers of computer architecture and a lead designer of the DEC Alpha microprocessor, wrote the following about the future of computer architecture research:

> Across the industry, today's chips are largely able to execute code faster than we can feed them with instructions and data. There are no longer performance bottlenecks in the floating-point multiplier or in having only a single integer unit. The real design action is in memory subsystems—caches, buses, bandwidth, and latency.
>
> An anecdote: in a recent database benchmark study using TPC-C, both 200-MHz Pentium Pro and 400-MHz 21164 Alpha systems were measured at 4.2-4.5 CPU cycles per instruction retired. In other words, three out of every four CPU cycles retired zero instructions: most were spent waiting for memory….Processor speed has seriously outstripped memory speed.
>
> Increasing the width of instruction issue and increasing the number of simultaneous instruction streams only makes the memory bottleneck worse. If a CPU chip today needs to move 2 GBytes/s (say, 16 bytes every 8 ns) across the pins to keep itself busy, imagine a chip in the foreseeable future with twice the clock rate, twice the issue width, and two instruction streams. All these factors multiply together to require about 16 GBytes/s of pin bandwidth to keep this chip busy. It is not clear whether pin bandwidth can keep up—32 bytes every 2 ns?
>
> I expect that over the coming decade memory subsystems design will be the *only* important design issue for microprocessors.[1]

Sites put it quite bluntly, even titling his article "It's the Memory, Stupid!" Because of the increasing processor and memory speed gap, the organization, architecture, and design of memory subsystems (particularly DRAM subsystems) have become dominant parts of computer system design.

Accordingly, memory system behavior has become a focal point of computer architecture research. Recent research ranges from studies of memory controller design[2,3] to attempts to

Published by the IEEE Computer Society

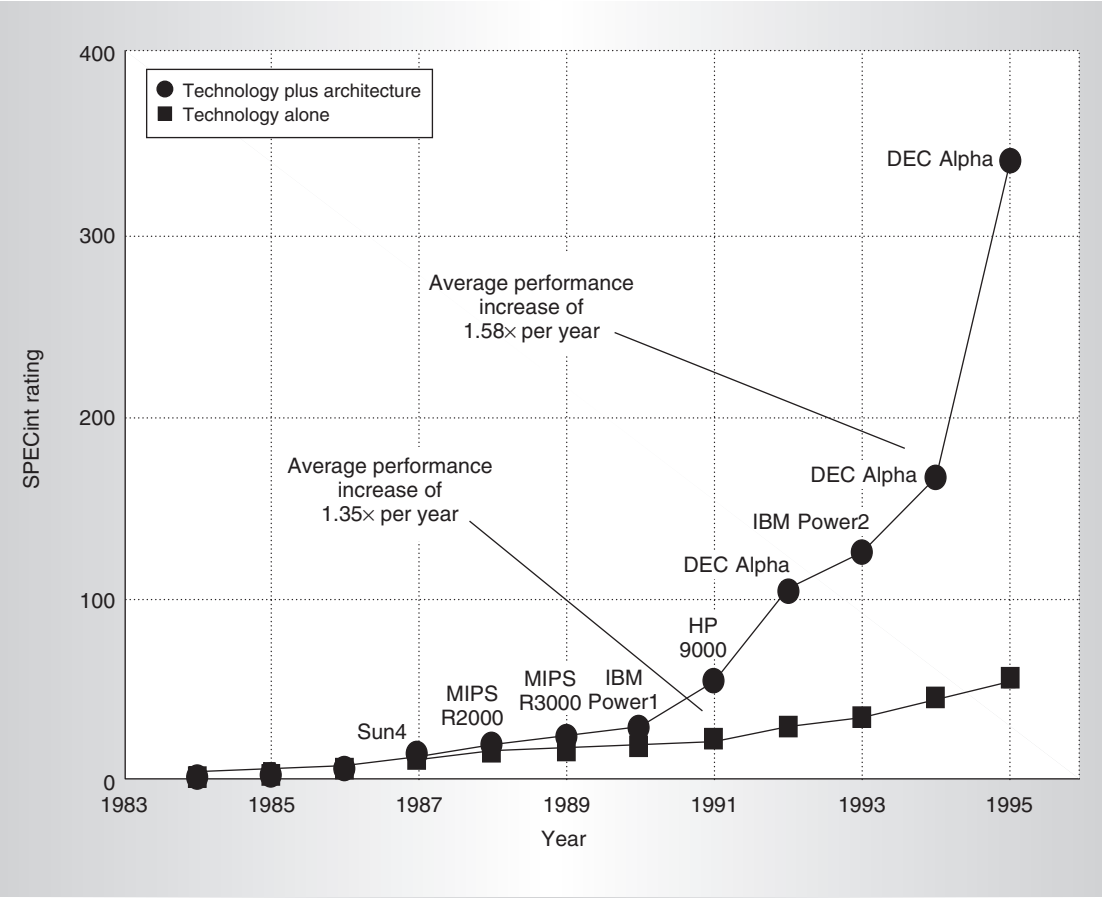

Figure 1. Relative importance of architecture and technology. Growth in microprocessor performance has been dramatic since the mid-1980s, averaging 50 to 60 percent a year. Earlier, microprocessor performance growth was largely technology driven and averaged about 35 percent a year. The difference is attributable to advances in computer architecture. (Source: J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd ed., Morgan Kaufmann, 1996)

integrate the DRAM core with the processor core for improved memory bandwidth and power consumption.[4-6] Other recent work looks at the performance of different commercial DRAMs at the architecture level and a wide range of DRAM bus and memory controller organizations at the system level.[7,8] Most work done at the DRAM architecture level—for example, evolving the DRAM interface from fast page mode (FPM) to extended data out (EDO) to SDRAM to double data rate (DDR) SDRAM—has aimed at improving the individual DRAM device's bandwidth. Simply improving bandwidth, however, does not necessarily imply a significant improvement in total execution time.

Figure 1 shows why paying attention to the architecture level is important, particularly concerning DRAM systems. Before the mid-1980s, technological advances largely drove microprocessor performance. Since then, microprocessors have relied on advances in both technology and architecture for improved performance. The result, by the mid-1990s, was a factor-of-five improvement over relying on technology alone.

One reason for the growing performance gap between processors and DRAM systems is that relatively few studies exist on the architecture of DRAM devices and systems. As a result, DRAMs have relatively little to draw on for performance improvements. Whereas processor performance has relied on improvements in both technology and architecture, DRAM performance has relied on technology improvements and only bandwidth-related

improvements in architecture. Further hobbling DRAM is the fact that memory technology improvements aim primarily at increasing DRAM capacity, not DRAM speed. So, whereas technology advances improve microprocessor performance 35 percent a year, they improve DRAM performance only 7 percent a year.[9] Clearly, the industry could regain some of this difference by focusing all technology improvements on DRAM performance rather than DRAM capacity. Nevertheless, DRAM performance would still trail microprocessor performance unless it drew from advances in both technology and architecture.

To meet the need for architecture- and system-level DRAM studies, my graduate students and I have developed a simulation framework that places disparate DRAM architectures on the same footing. We based the framework on a model that defines a continuum of design choices covering most contemporary DRAM architectures, such as Rambus, Direct Rambus, SDRAM, and DDR SDRAM. Using this framework, we investigated system-level parameters, including bus width, bus speed, number of independent channels, logical organization of channels, degree of banking, degree of interleaving, read burst width, write burst width, split-transaction versus pipelined buses, symmetric versus asymmetric read/write request shapes, and others. These are system-level parameters (as opposed to architecture-level) because they can be defined independently of the CPU or DRAM architectures.

This article presents the simulation framework and an initial study of system-level parameters, including bus speed, bus width, number of independent channels, degree of banking, and read/write burst width. Despite the large design space this study covers, it only begins to explore memory system organizations. We modeled a high-performance uniprocessor system—featuring a 2-GHz out-of-order superscalar CPU with lockup-free L1 and L2 caches—and used the more memory-intensive applications in the SPEC95 integer suite. This study asks and answers the following questions (clearly, our results and conclusions depend on our system configuration and choice of benchmarks):

- *How important are design choices at the DRAM system level?* The organization choices are extremely important. With the CPU architecture, L1/L2 cache organizations, DRAM architecture, and DRAM speed remaining constant, the choices at the organization level can affect total execution time by a factor of two. Memory system organization choices can affect memory overhead even more heavily, but most of this overhead is hidden behind program execution.
- *What system-level parameters most affect performance?* With other factors remaining constant, the read/write burst width can cause twofold differences in total execution time. The memory channel's cycle time can be responsible for a factor of two. The number of independent channels connecting the CPU to the DRAMs can effect a 25 percent performance change. Other parameters we studied are responsible for differences of less than 15 percent in total execution time. (The term *burst width* does not imply that the model is a burst-mode model. It refers to data access granularity; for example, direct Rambus has a packetized DRAM interface, rather than burst-mode DRAMs such as SDRAM or enhanced SDRAM (ESDRAM). However, its access granularity is 128 bits, or 16 bytes. Thus, we would model it as having a 16-byte burst width.)
- *What are the performance tradeoffs between the number of independent channels, the channel width, the channel speed, and the total system bandwidth (number of channels × channel width × channel speed)?* As you might guess, total per-channel bandwidth (bus width × bus speed) is often more important than either bus width or bus speed because, to the first order, it takes the same amount of time to send 128 bits down a 16-bit, 800-MHz channel as down a 128-bit, 100-MHz channel. However, with the appropriate burst size, the design space has large, relatively flat regions, indicating that giving up only modest performance (for example, 10 to 20 percent) can save significant costs (such as cutting the number of data pins in half or more and at the same time cutting the channel speed in half or more).

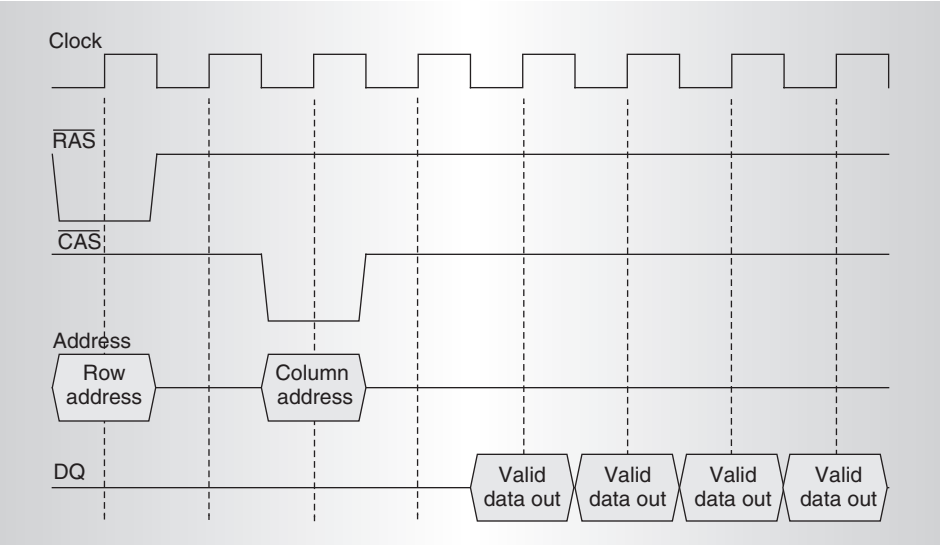

Figure 2. SDRAM read operation clock diagram. SDRAMs contain a writable register for the request length, allowing high-speed column access.

These are insights that you can discover only by studying DRAM issues at the system level. For example, the delay-locked loop (DLL) is present in DDR SDRAMs to align DRAM output with the global system clock and make the DRAMs in a single rank appear to have similar timing characteristics—that is, to deskew the output of devices in a module. The disadvantage is a per-device cost for the circuit in terms of both die size and power consumption. A single module-resident DLL or phase-locked loop (PLL) to align output with the global clock and a set of FIFOs at the memory controller to deskew between devices within a rank could achieve the same function. (Arguably, you could do a better job of deskewing at the memory controller, since deskewing at the DRAM puts the decision making in the hands of the DRAM and not the memory controller—and the controller is the reference point at which the amount of skew matters most.)

The tradeoff would be to increase the complexity of a single memory controller and add a single DLL/PLL to each module, in return for decreasing the complexity and power consumption of every DRAM in the system. This type of tradeoff can be quantified only by studying DRAM issues at the system level. Furthermore, as DRAM designs become increasingly complex, this type of tradeoff is likely to present itself more often.

Simulation framework and experimental methodology

The basic goal of our work is to define a primary memory system model that represents most existing DRAM organizations: burst-mode organizations such as JEDEC SDRAMs (including DDR and DDR2) and packetized organizations such as Rambus—the two main competing commercial standards—as well as almost everything between.

Device-level access timing

Most DRAMs in use today are synchronous: they run off an external clock derived from the bus. Figure 2 shows a typical SDRAM timing diagram. SDRAM devices typically contain a programmable register that holds a bytes-per-request value. SDRAMs can therefore return the bytes for a large request over several cycles. There is only one burst width; all reads and writes use the same transaction granularity.

The timing diagrams of all modern DRAMs look similar, the most noticeable differences being the clock frequencies and the number of cycles per transaction phase. For example, Figure 3 shows the read transaction timing diagram for the direct Rambus (DRDRAM), a radical departure from traditional DRAMs. The diagram shows the earliest point at which the active bank can be precharged and reactivated, although other

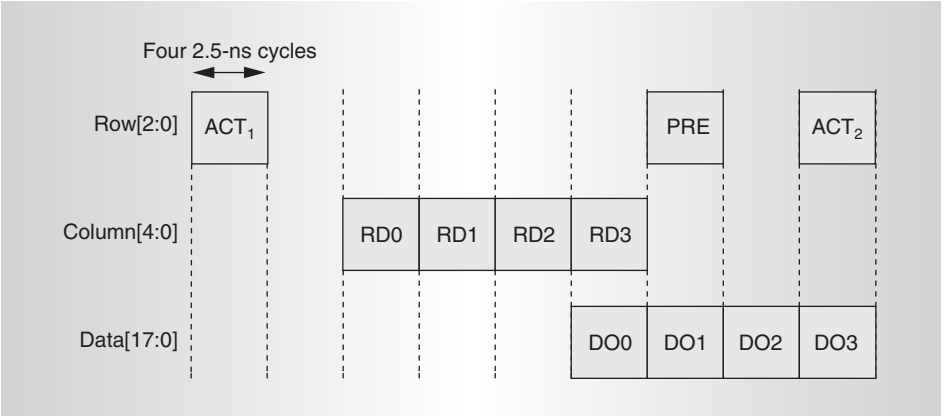

Figure 3. Direct Rambus read clock diagram. Direct Rambus DRAMs transfer on both edges of a fast clock and can handle multiple simultaneous requests to different banks.

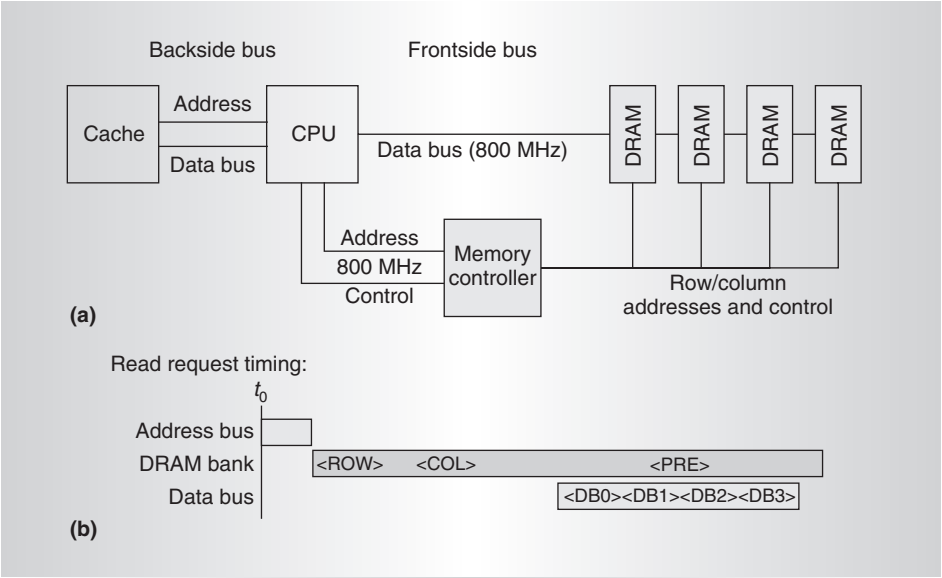

Figure 4. Typical uniprocessor system organization: architecture (a) and read request timing diagram (b). This is the system modeled in this article. Note the absence of an accelerated graphics port, which would overestimate system performance. (Source: R.C. Schumann[10])

banks can be activated sooner. ACT1 and ACT2 are row activate commands, PRE is the precharge command, RD0 to RD3 are read commands, and DO0 to DO3 are corresponding data-out timings.

DRDRAM uses a 400-MHz, 3-byte channel (2 bytes for data, 1 for addresses and commands). DRDRAM parts transfer on both clock edges, implying a maximum bandwidth of 1.6 Gbytes/s. Because DRDRAM partitions the bus into different components, four transactions can simultaneously utilize the different portions of the DRDRAM interface (the overlap is four, not three, because control information can be embedded in column address packets).

System-level access timing

At the system level, request timing is slightly different. In PC and most workstation architectures, like the one shown in Figure 4a, an external memory controller sends row and column requests directly to the DRAMs. This is a relatively simple architecture in which the DRAM output pins connect directly to the CPU data pins (an example is the Alpha serv-

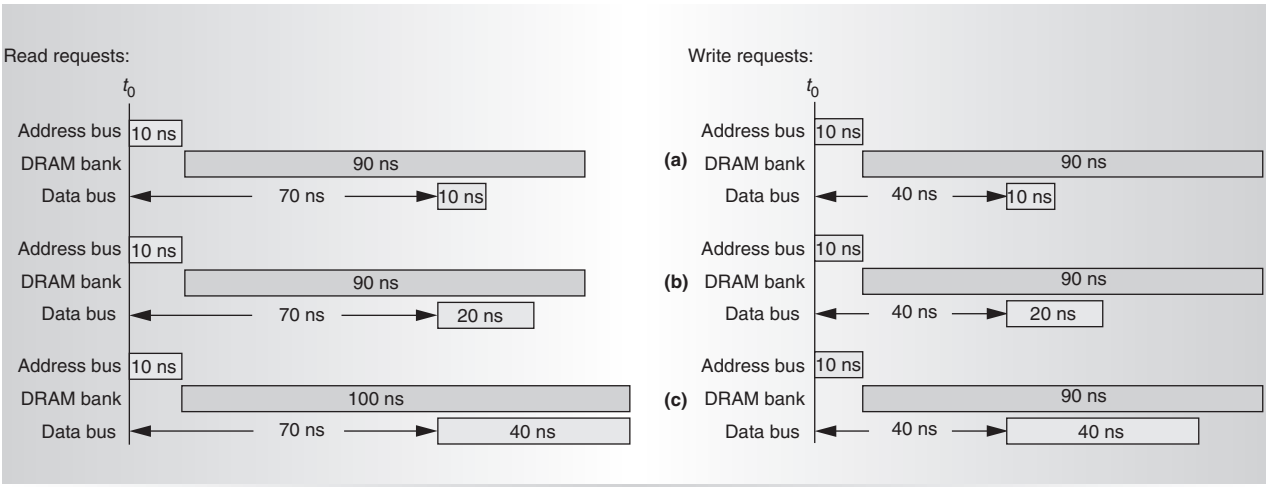

Figure 5. Bus and bank occupancies for 100-MHz channel: burst-widths equal to the bus width (a), two times the bus width (b), and four times the bus width (c). Each DRAM request requires the address bus, the data bus, and the bank for which the request is destined. Note that the timings are not fixed; queuing and scheduling by the memory controller can delay a request's timing.

er described by Schumann[10]). This modularity makes more CPU pins available for data (as opposed to putting the memory controller on the CPU), thereby increasing the CPU's potential pin bandwidth. Modularity also allows simpler implementation of multiprocessor systems (including uniprocessor systems with external graphics processors) by providing a single contact point for the DRAM system. It provides upward compatibility by allowing the system to use future DRAMs without significant redesign (only the lowest-level memory controller needs to be redesigned). Finally, modularity is essential in server configurations with large amounts of memory in large numbers of dual in-line memory modules because such a bus system's capacitance usually requires that the DIMMs be placed on separate, multiplexed, buses.[10]

As a side effect of the modular design, the operation timing changes slightly. The most obvious change is that the CPU sends the address to the memory controller all at once, and not until the following cycle does the row-address command activate the DRAM bank. Figure 4b shows how SDRAM-like access timing fits within the framework of a full DRAM system.

For the DRAM core speed, we use parameters from the latest SDRAM (as of this writing), which has reasonably fast timing specifications and is common to PC-100 and direct Rambus designs. This core speed gives us read and write bus, and bank occupancies similar to those reported in the literature.[10,11] Figure 5 illustrates the bus and bank occupancies for a 100-MHz channel. The figure presents burst widths equal to the data bus width, twice the bus width, and four times the bus width. A burst is the smallest atomic transaction size—all read and write requests are processed as an integral number of bursts, and the bursts of different requests can be multiplexed in time over the same channel.

This interface model covers burst-mode DRAM architectures such as SDRAM, ESDRAM, and burst-mode synchronous-link DRAM (SLDRAM); it also covers packetized DRAM architectures such as Rambus, direct Rambus, and packetized SLDRAM. The only difference in moving to a packetized interface is that the address bus packet scales with the data bus packet in the length of time it occupies the address bus. Since the two are scheduled together, this scheme imposes no additional overhead.

Channels and banks

A single DRAM device can handle one request at a time and produces a certain number of bits per request; this number is the device-level transfer width. DRAM devices are grouped into independent banks, each of which can service a different request than all other banks are servicing at any given

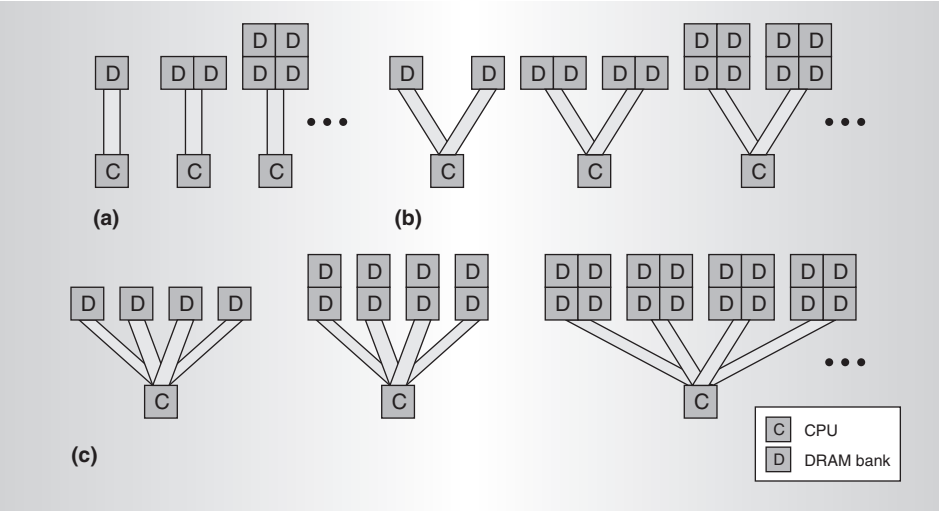

Figure 6. Channels and banks: one (a), two (b), and four (c) independent channels, banking degrees of 1, 2, 4, and so on.

moment. The bank is the smallest unit of granularity in this model. Whether a bank is a collection of devices, a single physical device, or a subcomponent of a single physical device need not be specified. Figure 6 shows several example memory system organizations that our model can represent. In the figure, we vary the number of independent channels and the number of independent DRAM banks attached to each channel.

A single bank has a transfer width at least as wide as the data bus. Each channel is a split-transaction address-bus and data-bus pair, and connects to potentially multiple banks, each operated independently. Using multiple banks per channel supports concurrent transactions at the channel level. The CPU connects through an on-board memory controller to potentially multiple channels, each operated independently. Using multiple channels supports concurrent transactions at the DRAM subsystem level. Bit mapping from address to channel, bank, and row attempts to best exploit the physical organization's available concurrency by assigning the lowest-order bits (which change the most frequently) to the channel number, the next bits to the bank number, and so on. Counters in our simulator show that requests are divided evenly across a system's channels and across each channel's banks.

This very simple organization accounts for most existing DRAM architectures. Clearly, it can emulate organizations such as SDRAM and DDR SDRAM, but by increasing the degree of banking and scaling the channel width and speed, it can also do a fair job of emulating Rambus-style organizations that are highly banked internally.

Concurrency within a single channel

With multiple channels, it is easy to see that a system can exploit concurrency. However, with sufficient banking, a single channel can also support concurrency. Figure 7 illustrates how back-to-back requests can overlap in time on a split-transaction bus.[11] Back-to-back reads and back-to-back writes can be pipelined, provided they require different banks. Back-to-back read/write pairs can be similarly pipelined, though less efficiently. But under the right conditions, it is also possible to nestle writes inside of reads, as Figure 7b shows. This feature is possible only because of the asymmetric nature of read/write requests, and it is possible only on a split-transaction bus. Although reads and writes are asymmetric, they look less so as the burst width increases and the data bus turnaround time increases. We model bus turnaround time as a constant number of bus cycles; for this study, we used a one-cycle delay for all buses.

Burst ordering

If a burst is smaller than the level-2 cache line size, we have four options for ordering the

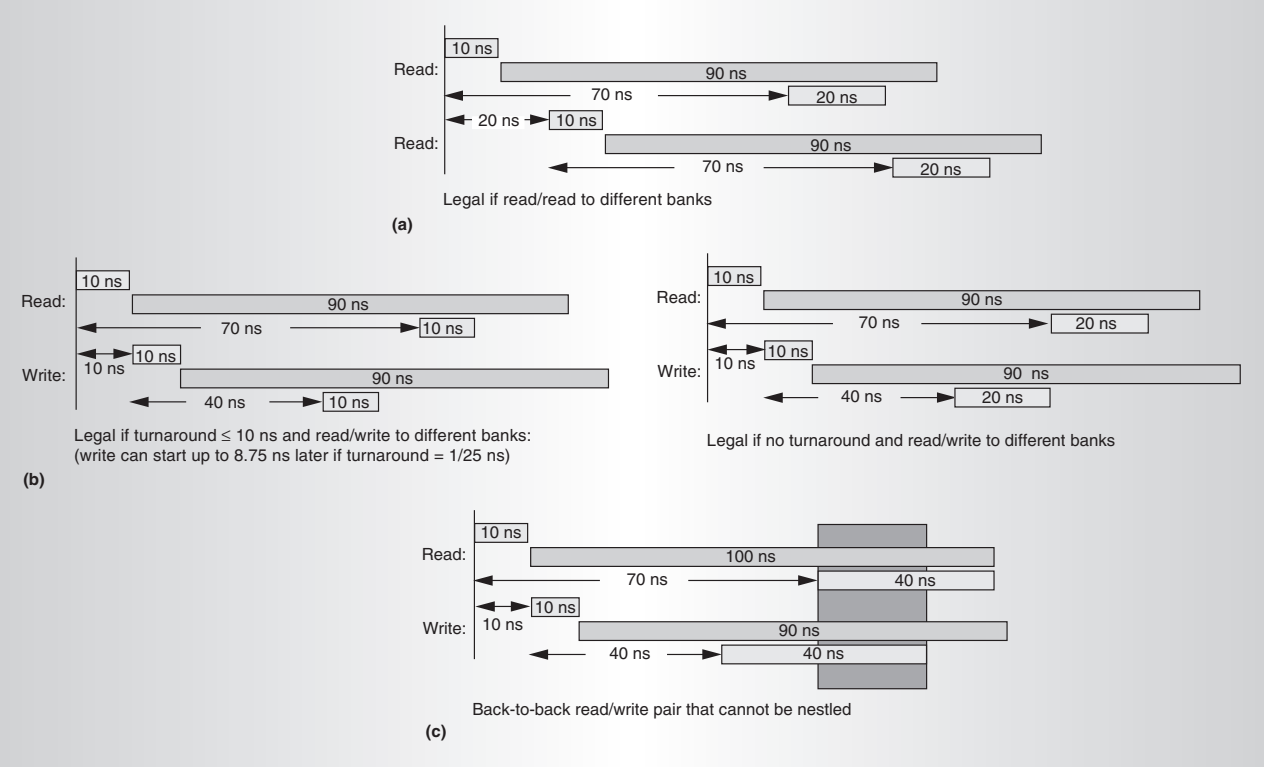

Figure 7. Concurrency within a single channel. If two concurrent reads require different banks, they can be pipelined across the address and data bus (a). Writes can be nestled inside of reads, provided the bus turnaround time is low and the burst width is small (b). However, for some burst sizes, reads and writes cannot be nestled (c).

burst-sized blocks that make up the request. These options, illustrated in Figure 8, are the following multiplexing schemes:

- *One request, one burst*, in which the burst size is equal to the cache line size. In this scenario, two requests that require the same resource at the same time (either a channel or a bank within a channel) must go one after the other.
- *Critical block first*, in which the burst block containing the critical word is fetched ahead of the rest of the cache line, but two simultaneous requests are still processed one after the other. Two requests A and B that require the same bank and arrive at the DRAM system simultaneously are processed in the order A′A″B′B″, where X′ is the critical burst of cache block X, and X″ is the remainder of the cache block.
- *Minimally interleaved*, in which the burst block containing the critical word has a higher priority than the noncritical burst blocks of other transactions. Two requests A and B that require the same bank and arrive at the DRAM system simultaneously are processed in the order A′B′A″B″.
- *Round-robin*, which processes two simultaneous requests in the order $A'B'A_1''B_1''A_2''B_2'' \ldots A_{n-1}''B_{n-1}''$, where X_i'' represents the ith burst of the remaining cache line (excluding the critical burst).

An obvious question is how many buffers are needed to handle each of these schemes; if writes can always be preempted, they must be queued indefinitely. How extensively would a real application take advantage of this? In this study, we use the round-robin approach: The block containing the critical word is always fetched first and takes priority over any other block in the queue, unless that block also contains a critical word. We always give write requests the lowest priority, and they stack up in the queue until all the reads drain from it.

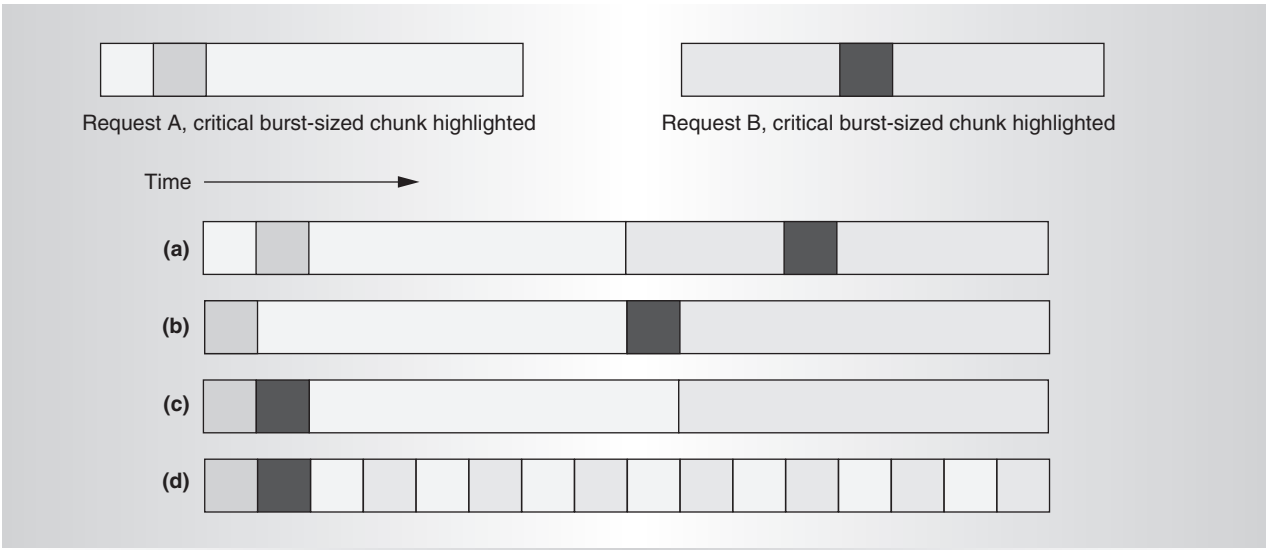

Figure 8. Time division multiplexing of simultaneous read requests. For simultaneous read requests that require the same resource, there are at least four interleaving schemes: one request, one burst (a); critical block first (b); minimally interleaved (c); and round-robin (d).

Bit addressing and page policy

We choose bit assignments to exploit page mode and maximize the degree of memory concurrency achieved by the application. To maximize the page mode effectiveness, we divide memory into DRAM-page-sized chunks. To maximize request concurrency, the lowest-order bits after the DRAM page offset choose the DRAM channel, the next bits choose the bank, and the highest-order bits choose the row.

We assign address bits so that the most significant bits identify the system's smallest-scale component, and the least significant, which should change most often from request to request, identify the system's largest-scale component. Simultaneous requests to adjacent DRAM-page-sized blocks in the memory system go out on two different DRAM channels if available, and the requests that make up a cache block fill go to the same DRAM page. This organization exploits concurrency to the greatest degree possible because sequential addresses are striped across entire DRAM channels, and the requests that make up a cache block fill can exploit a DRAM's page mode.

We assume a close-page auto-precharge policy[12] in our simulations, and we assume SRAM buffering of one DRAM page in ESDRAM style. The DDR2 working group is leaning toward a close-page auto-precharge policy. This makes sense as systems go to larger amounts of DRAM; to fully exploit an open-page mode, a memory controller must retain information on every open page in the DRAM system, which can become expensive. Our address-bit assignments direct adjacent cache fill requests to the same DRAM page; in a normal close-page policy, these requests would encounter an intervening precharge cycle. Nonetheless, the simulation doesn't stall in this situation. Using a DRAM's page mode is possible only because we expect the DRAM device to have ESDRAM-style page buffering: one full page of SRAM storage per internal bank. As a result of such buffering, a memory controller can implement a close-page auto-precharge policy without destroying any read locality for future requests to the same DRAM page. The penalty is that the memory controller must keep track of all open pages.

CPU model

To obtain accurate memory request timing in a dynamically reordered instruction stream, we integrated our code into SimpleScalar 3.0a, an execution-driven simulator of an aggressive out-of-order processor. Our simulated processor is eight-way superscalar; its simulated cycle time is 0.5 ns (a 2-GHz clock). Its L1 caches are split into 64 Kbytes and 64 Kbytes; both are two-way set associative, and

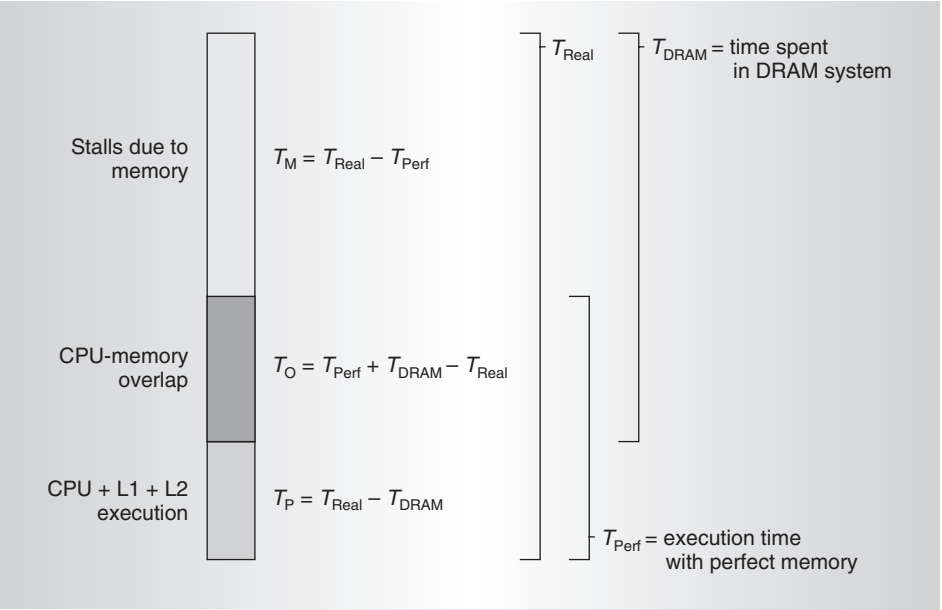

Figure 9. Definitions of execution time breakdowns. The results of several simulations show time spent in the memory system versus time spent processing versus the amount of memory latency hidden by the CPU.

both have 64-byte line sizes. Its L2 cache is a unified 1-Mbyte, four-way set-associative, write-back cache with a 128-byte line size and a 10-cycle access time. The L1 and L2 caches are both lockup-free, and both allow up to 32 outstanding requests at a time. The degree of concurrency allowed by the bus and memory system depends on the configuration (the number of independent channels, the number of banks per channel, and so on).

For our lockup-free cache model, a load instruction that misses the L2 cache is blocked until it obtains a miss status holding register (MSHR), which it holds only until the critical burst of data returns (remember that the atomic unit of transfer between the CPU and DRAM system is a burst, which is variable-sized in our framework). This scheme frees the MSHR quickly, allowing subsequent load instructions that miss the L2 cache to commence as soon as possible. The scheme is relatively expensive to implement because it assumes that cache tags can be checked for subsequently arriving blocks of data without disturbing regular cache traffic. We model this optimization to put the highest possible pressure on the physical memory system—it represents the highest rate at which the processor can generate concurrent memory accesses given the number of available MSHRs.

Timing calculations

Much of the DRAM access time overlaps instruction execution. To determine the degree of overlap, we run a second simulation with perfect primary memory (no overhead). Using a methodology similar to that described by Cuppu and Jacob,[7] we partition the total application execution time into three components: T_P, time spent processing; T_M, time spent stalling for memory; and T_O, the portion of time spent in the memory system that successfully overlaps processor execution. Here, time spent processing includes all activity above the primary memory system; that is, it contains all processor execution time and L1 and L2 cache activity. Let T_{Real} be the total execution time for the realistic simulation, let T_{Perf} be the execution time with a perfect DRAM system, and let T_{DRAM} be the total time spent in the DRAM system. Then we have the following:

- $T_P = T_{Real} - T_{DRAM}$
- $T_M = T_{Real} - T_{Perf}$
- $T_O = T_{Perf} + T_{DRAM} - T_{Real}$

Figure 9 illustrates the relationships among these time parameters.

Experimental results

The simulations in our study cover most of the space defined by the cross-product of these variables:

- 1, 2, and 4 independent channels;
- 1, 2, and 4 banks per channel;
- 8-, 16-, 32-, 64-, and 128-byte burst widths;
- 1-, 2-, 4-, and 8-byte data bus widths;
- 100-, 200-, 400-, and 800-MHz bus speeds; and
- the gcc and perl benchmarks from SPEC, known to have relatively large memory footprints.

The simulated L1 and L2 cache line sizes are 64 and 128 bytes. The unit of performance is cycles per instruction (CPI), a direct measurement of execution time, given a fixed cycle time and the length of each program. For some system configurations, we further break down total execution time into the timing components described in the preceding subsection.

Figure 10 shows our simulation results, with total execution time as a function of both burst width and memory system bandwidth. On the *x*-axis is system bandwidth, which is total channels × channel width × channel speed. For each bandwidth value, several configurations represent different combinations of channels, width, and speed. For each configuration, five bars represent total execution time for burst widths of 8, 16, 32, 64, and 128 bytes. For perl, Figure 10a shows all possible bandwidth configurations—although the graph is cluttered, it gives a distinct feel for the complete design space. For gcc, Figure 10b shows all configurations at 800 MHz and omits the other configurations for clarity.

Among other things, the graphs show that for a given bandwidth configuration, the choice of burst size can affect execution time significantly—for example, by a factor of two for perl and 1.5 for gcc. This shows the importance of selecting an appropriate burst size. Although optimal burst width depends on bandwidth and channel speed (optimal burst width is around 32 bytes for 200-MHz channels and around 64 bytes for 400- and 800-MHz channels), it generally tends to be relatively large; for most configurations, it is 64 bytes. Other experiments of ours show that the optimal burst size also depends on cache block size—in general, the burst should be large enough to fetch a level-2 cache block in two requests.

In Figure 10, if you look only at the 64-byte burst widths (the optimal points, for the most part), you see a gradual curve that slopes down as bandwidth increases, showing the effects of increased bandwidth on execution time. Once the system bandwidth is sufficient for the application in question, the slope reflects a mere 10 percent or less improvement in execution time for every doubling of memory system bandwidth, which is far less significant than burst width's effect on performance. Within a fixed bandwidth class, the choice of bus speed and number of channels is significant, but not as significant as doubling or halving the bandwidth. For example, at 800 Mbytes/s, moving from a quad 200-MHz 1-byte bus organization to a dual 400-MHz 1-byte bus organization to a single 800-MHz 1-byte bus organization yields a smaller performance difference than moving to a 400-Mbyte/s or 1.6-Gbyte/s organization.

Overall, our results show that burst width is an extremely significant parameter that overshadows both raw bandwidth and the details of how we choose the bandwidth (number of channels, channel width, and channel speed). Moreover, completely acceptable performance is possible with slow, multichannel buses, suggesting that even single-processor applications can exploit concurrency in the memory system.

Memory system organization is extremely important and can significantly affect an application's total execution. Unfortunately, no memory system design choices are universally good. The only rules of thumb are

- the optimal burst size scales with the L2 block size, and
- faster channels are slightly better than slower channels, given the same system bandwidth.

However, the second rule of thumb is probably due to our modeling the bus turnaround time as a constant number of bus cycles, rather than a constant number of nanoseconds; this translates to a lower overhead for buses with shorter cycle times. Because the interactions between

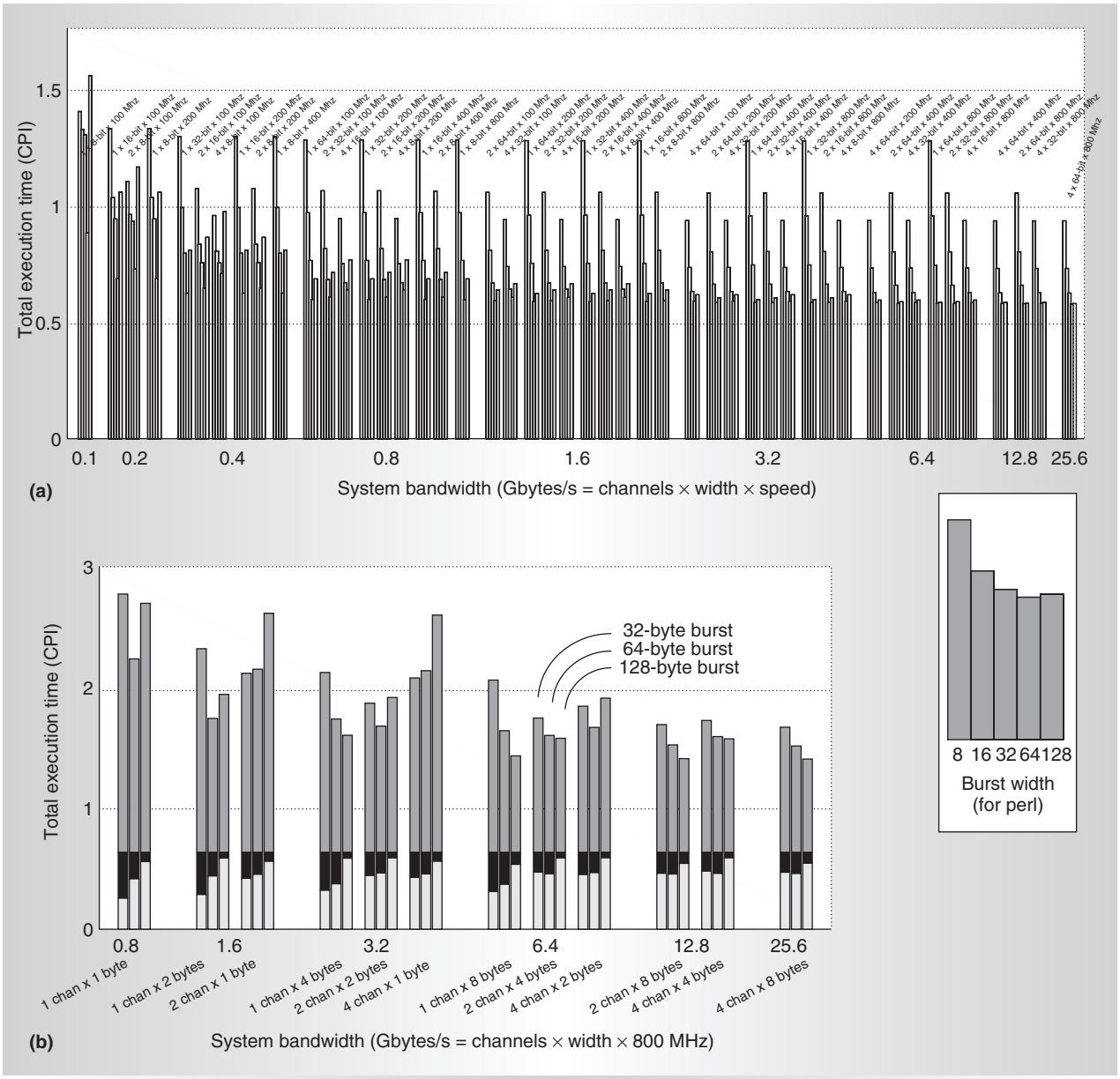

Figure 10. Simulation bandwidths and burst widths: SPEC perl, four banks per channel (a); SPEC gcc, two banks per channel (b).

system configuration and burst size can affect system performance by such a large factor (two times or more), designing the entire memory system to fit together is critically important—no one component can be optimized in isolation. Given that the optimal burst width scales with the level-2 cache block size, even the cache organization must play a role in the design of the primary memory system.

As mentioned earlier, however, once the appropriate burst size is chosen, the design space has large, relatively flat regions, indicating that giving up only modest performance (for example, 10 to 20 percent) will save significant costs (for example, cutting the number of data pins in half or more and at the same time cutting the channel speed in half or more). Again, these are insights we can discover only by studying DRAM issues at the system level. The implication: Compared with the present focus on maximizing per-device bandwidth by increasing data pin toggle rates, system-level approaches to higher DRAM system performance can yield as good or better

results with significantly less cost and engineering effort. MICRO

Acknowledgments

Two of my students, Vinodh Cuppu and Ken Powers, collected the data presented in this article. Given that the data represents a bit more than six months of simulations, I certainly could not have written the article without their help.

Vinodh Cuppu was supported in part by NSF grant EIA-9806645 and NSF Career Award CCR-9983618. Ken Powers was supported in part by NSF's sponsorship of undergraduate research through grant NSF-9912218. I am supported in part by NSF Career Award CCR-9983618, NSF grant EIA-9806645, NSF grant EIA-0000439, and DoD award AFOSR-F496200110374, as well as by Compaq and IBM.

References

1. R. Sites, "It's the Memory, Stupid!" *Microprocessor Report*, vol. 10, no. 10, Aug. 1996, pp. 1, 2.
2. S.A. McKee and W.A. Wulf, "Access Ordering and Memory-Conscious Cache Utilization," *Proc. 1st IEEE Symp. High-Performance Computer Architecture* (HPCA-1 95), IEEE CS Press, 1995, pp. 253-262.
3. J. Carter et al., "Impulse: Building a Smarter Memory Controller," *Proc. 5th Int'l Conf. High-Performance Computer Architecture* (HPCA-5 99), IEEE CS Press, 1999, pp. 70-79.
4. D. Burger, J.R. Goodman, and A. Kagi, "Memory Bandwidth Limitations of Future Microprocessors," *Proc. 23rd Int'l Symp. Computer Architecture* (ISCA 96), ACM Press, 1996, pp. 78-89.
5. A. Saulsbury, F. Pong, and A. Nowatzyk, "Missing the Memory Wall: The Case for Processor/Memory Integration," *Proc. 23rd Int'l Symp. Computer Architecture* (ISCA 96), ACM Press, 1996, pp. 90-101.
6. C. Kozyrakis et al., "Scalable Processors in the Billion-Transistor Era: IRAM," *Computer*, vol. 30, no. 9, Sept. 1997, pp. 75-78.
7. V. Cuppu and B. Jacob, "Concurrency, Latency, or System Overhead: Which Has the Largest Impact on Uniprocessor DRAM System Performance?" *Proc. 28th Int'l Symp. Computer Architecture* (ISCA 01), IEEE CS Press, 2001, pp. 62-71.
8. V. Cuppu et al., "High Performance DRAMs in Workstation Environments," *IEEE Trans. Computers*, vol. 50, no. 11, Nov. 2001, pp. 1133-1153.
9. W.A. Wulf and S.A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *ACM Computer Architecture News*, vol. 23, no. 1, Mar. 1995, pp. 20-24.
10. R.C. Schumann, "Design of the 21174 Memory Controller for Digital Personal Workstations," *Digital Technical J.*, vol. 9, no. 2, 1997, pp. 57-70.
11. W.R. Bryg, K.K. Chan, and N.S. Fiduccia, "A High-Performance, Low-Cost Multiprocessor Bus for Workstations and Midrange Servers," *Hewlett-Packard J.*, vol. 47, no. 1, Feb. 1996.
12. B. Davis et al., "DDR2 and Low Latency Variants," *Proc. Memory Wall Workshop at the 26th Int'l Symp. Computer Architecture*, 2000; http://www.ece.mtu.edu/faculty/btdavis/papers/mem_wall.isca2k.pdf.

Bruce Jacob is an associate professor in the Electrical and Computer Engineering Department of the University of Maryland, College Park. His research interests include memory systems, embedded systems, and operating systems. Jacob has a PhD in computer science and engineering from the University of Michigan, Ann Arbor. He is a member of the IEEE and the ACM.

Direct questions and comments about this article to Bruce Jacob, Dept. of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742; blj@eng.umd.edu.