Cached DRAM for ILP Processor Memory Access Latency Reduction

Cached DRAM adds a small cache onto a DRAM chip to reduce average DRAM access latency. The authors compare cached DRAM with other advanced DRAM techniques for reducing memory access latency in instruction-level-parallelism processors.

Zhao Zhang, Zhichun Zhu, and Xiaodong Zhang
College of William and Mary

As the speed gap between processor and memory widens, data-intensive applications such as commercial workloads increase demands on main memory systems. Consequently, memory stall time—both latency time and bandwidth time—can increase dramatically, significantly impeding the performance of these applications. DRAM latency, or the minimum time for a DRAM to physically read or write data, mainly determines latency time. The data transfer rate through the memory bus determines bandwidth time. Burger, Goodman, and Kägi show that memory bandwidth is a major performance bottleneck in memory systems.[1] More recently, Cuppu et al. indicate that with improvements in bus technology, the most advanced memory systems, such as synchronous DRAM (SDRAM), enhanced SDRAM, and Rambus DRAM, have significantly reduced bandwidth time.[2] However, DRAM speed has improved little. DRAM speed is a major factor in determining memory stall time, which significantly affects the performance of data-intensive applications such as commercial workloads.

In a cached DRAM, a small or on-memory cache is added onto the DRAM core. The on-memory cache exploits the locality that appears on the main memory side. The DRAM core can transfer a large block of data to the on-memory cache in one DRAM cycle. This data block can be several dozen times larger than an L2 cache line. The on-memory cache takes advantage of the DRAM chip's high internal bandwidth, which can be as high as a few hundred gigabytes per second.

Hsu and Smith classify cached DRAM organizations into two groups: those where the on-memory cache contains only a single large line buffering an entire row of the memory array, and those where the on-memory cache contains multiple regular data cache lines organized as direct-mapped or set-associative structures.[3] In a third class combining these two forms, the on-memory cache contains multiple large cache lines buffering multiple rows of the memory array organized as direct-mapped or set-associative structures. Our work and other related studies belong to this third class.[4-5]

Cached DRAM improves memory access efficiency for technical workloads on a relatively simple processor model with small data caches (and in some cases, even without data caches).[4-7] In a modern computer system, the CPU is a complex instruction-level parallelism

0272-1732/01/$10.00 © 2001 IEEE

(ILP) processor, and caches are hierarchical and large. Thus, a cached DRAM's architectural context has dramatically changed and evolved. Koganti and Kedem investigated the performance potential of cached DRAM in systems with ILP processors and found it effective.[4] To further investigate the ILP effects and compare cached DRAM with other advanced DRAM organizations and interleaving techniques, we studied cached DRAM in the context of processors with full ILP capabilities and large data caches.

Conducting simulation-based experiments, we compare cached DRAM with several commercially available DRAM schemes. Our results show that the cached DRAM outperforms other DRAM architectures for these applications. Cached DRAM is not only effective with simple processors but also with modern ILP processors. Our focus is on further investigating the ILP effects.

DRAM background

Basic DRAM structure is a memory cell array, where each cell stores one bit of data as the charge on a single capacitor. Compared with static RAMs, DRAMs have several technical limitations.[8] The DRAM simple cell structure with one capacitor and one transistor makes the row access latency longer than that of SRAMs, which use multiple transistors to facilitate a cell. Also, the original signals in DRAM cells are destructive during read operations. The signals must be written back to the selected memory cells. In contrast, the signals in SRAM cells restore themselves after read operations. Finally, each DRAM cell must be refreshed periodically to charge the capacitor. In contrast, SRAMs hold their data bits using flip-flop gate circuits, retaining memory content as long as the power is on.

SRAMs are fast but expensive due to their low density. DRAMs are relatively slow but offer high density and low cost. Designers have widely used DRAMs to construct the main memory for most types of computer systems. The only exceptions are some vector computer systems that use SRAMs. All contemporary workstations, multiprocessor servers, and PCs use DRAMs for main memory modules and SRAMs for cache construction.

DRAM access consists of row and column access stages. During row access, a row (also called a page) containing the desired data is loaded into the row buffer. During column access, data is read or written according to its column address. Repeatedly accessing the data in the same row buffer only requires column access. However, if the next access goes to a different row in the memory array, the DRAM bank must be precharged before the next access. Periodically reading and writing back each row refreshes the DRAM bank. The DRAM is not accessible during precharge or refresh. The access time of a request is not a constant. It depends on whether the access is a page hit, if a precharge is needed, or if the DRAM is being refreshed.

Several recent commercial DRAM variants can reduce latency and/or improve the data transfer rate.

- *SDRAM.* The data access operations are synchronized with the processor under an external clock. This variant supports burst mode data access that reads or writes continuously allocated data blocks in the same row sequentially without idle intervals. SDRAMs normally have two or four independent data banks, providing an opportunity to overlap concurrent data accesses.
- *Enhanced SDRAM.* A single-line SRAM cache is integrated with each SDRAM memory bank's row buffer. If an access is a hit to the buffer, the access time is equivalent to that of accessing the fast SRAM cache. ESDRAM can overlap memory precharging and refreshing operations with cache accesses.
- *Rambus DRAM.* A high-speed but narrow (1-byte wide) bus connects the processor and multiple memory banks. This bus is multiplexed for transferring addresses and data. Both edges of the bus clock signal transfer data to double the data transfer rate. RDRAM memory banks can be independently accessed in a pipelining mode. Currently, RDRAMs support 8 or 16 banks.
- *Direct Rambus DRAM.* This is an improved version of RDRAM that provides a 1-byte wide address bus and a 2-byte wide data bus. Currently, DRDRAMs support 16 or 32 memory banks. Adjacent banks share half-page row buffers.

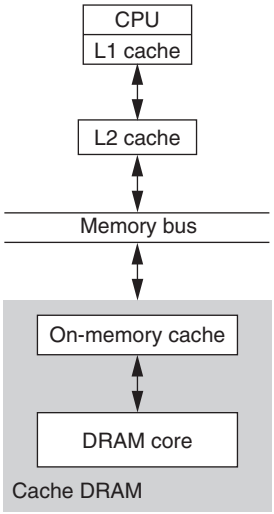

Figure 1. The lower box shows the cached DRAM where an on-memory cache connecting with the DRAM core interfaces with the memory bus.

Cached DRAM adds a small SRAM cache onto the DRAM core. It takes advantage of the huge internal bandwidth existing in the DRAM core, so that the cache block can be as large as a page. The processor usually has large caches, but the cache block is much smaller because of the limited bandwidth between the processor and the main memory. In general, a small cache with large block size can have a comparable miss rate to a large cache with a small block size.[7] The cached DRAM has a higher cache hit rate than other DRAM architectures because of its fully associative cache organization. Figure 1 shows the general concept of cached DRAM.

Contemporary processors aggressively exploit ILP by using superscalar, out-of-order execution, branch prediction, and nonblocking caches. As a result, the processor may issue multiple memory requests before the previous request is finished. Although the processor can keep running before the memory requests are finished, its ability to tolerate long memory latency is limited. Cached DRAM can provide fast response for a single memory request, and pipeline multiple requests to achieve high throughput.

Cached-DRAM operations

A cached DRAM is an integrated memory unit consisting of an on-memory cache and a DRAM core. Inside the cached DRAM, a wide internal bus connects the on-memory cache to the DRAM core.

The processor sends memory requests to the memory controller when an L2 cache miss happens. To take advantage of the low access latency of the on-memory cache, the memory controller maintains the on-memory cache tag and compares each tag with the tag portion of the address for every memory request. For each on-memory cache block, the memory controller also maintains the dirty flag, which indicates whether the block has been modified after it is loaded from the DRAM core.

For on-memory cache, a memory request is one of four types. In a *read hit*, the memory controller sends the on-memory cache a read command along with the block index and the column address via the address/command bus. This takes one bus cycle. The row address is not needed because the DRAM core is not accessed. The on-memory cache outputs the data in one bus cycle and sends it back to the memory controller and processor after another bus cycle. If the memory controller receives consecutive memory requests that are read hits, it processes them in a pipelined mode.

A *read miss* takes two processing steps. First, the row in the DRAM core that contains the data is read and transferred to the on-memory cache. Next, the data is read from the on-memory cache as if it were a read hit (a read miss can be overlapped with read hits). For the first step, the memory controller sends the DRAM a read command along with the row address and the replaced block index on the command/address bus. This step activates the row access of the DRAM core. The memory controller uses a modified least recently used (LRU) policy to find the block for a replacement.

A *write hit* employs a modified write-back policy in which data is only written into the on-memory cache. The memory controller sends the write command along with the block index and the column address onto the address/command bus, and sends the data onto the data bus. At the same time, the dirty flag is set for that block on the memory controller. The row address is not needed, because the DRAM core is not accessed. The processing of each write hit in a sequence can overlap with one another and with the processing of read requests.

In a *write miss*, the memory controller uses the modified LRU policy to select a block for a replacement. The write-allocate policy is enhanced with two steps. The memory controller sends the DRAM read command along with the block index and the row address onto the address/command bus. The row in the DRAM core that contains the writing address is first read from the DRAM core and transferred to the on-memory cache. The next step writes the data into the on-memory cache, operated as a write hit. These two steps can overlap with other read or write requests.

Table 1 shows an example of pipelining three continual read hits. Table 2 shows an example of a read miss that is overlapped with two read hits. The pipelining operations of write hits and write misses are similar.

The replacement policy of on-memory cache is a modified LRU policy that avoids

choosing a dirty block for replacement. If a dirty block were chosen for replacement, the block would have to be written back into the DRAM core first and would increase the latency of the memory request that causes the replacement. To increase the number of clean blocks available for replacement, the memory controller will schedule write-back requests for the dirty cache blocks as soon as the DRAM core is not busy.

ILP processors do not stall for a single L2 miss, so they may issue more memory requests when a previous request is processing. An outstanding read request may prevent dependent instructions in the instruction window from being issued to execution units, which is likely to reduce the ILP or fill the instruction window. On the other hand, outstanding writes do not influence ILP processors as long as the write buffer is not full. Therefore, the memory controller should schedule read requests prior to write requests.

Table 1. Example of pipelining three continual read hits (R1, R2, and R3).*

	Bus cycle								
Location	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Address/command bus	R1		R2		R3				
Cached DRAM data			D1	D1	D2	D2	D3	D3	
Processor data				D1	D1	D2	D2	D3	D3

*An R1 on the address/command bus indicates that the first read's block index and column address are sent on the address/command bus. A D1 on the cached DRAM data indicates that a block of data for the first read is available in the cached DRAM. A D1 on the processor data means that a block of data for the first read is available for the processor. R2 and R3 correspond to the second and third read commands/addresses. D2 and D3 correspond to the data items for the second and third reads.

Table 2. Example of pipelining a read miss (R1) and two read hits (R2 and R3).*

	Bus cycle									
Location	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Address/command bus	R1	R2		R1		R3				
Cached DRAM data				D2	D2	D1	D1	D3	D3	
Processor data					D2	D2	D1	D1	D3	D3

*The first R1 in the address/command bus indicates that the DRAM read command, the row address, and the block index are sent on the address/command bus. The second R1 indicates that a cache read command, the column address, and the block index are sent; R2 and R3 are signals of the two read hits on the bus. D1, D2, and D3 on the cached DRAM data indicate that the data for the reads are available on the cached DRAM. D1, D2, and D3 on the processor data mean the data for these reads are available for the processor.

Experimental environment

We used SPEC95 and TPC-C benchmarks as the workloads, and SimpleScalar as the base simulator.[9] We used the PostgreSQL (version 6.5) database system to support the TPC-C workload. This is the most advanced open source database system for basic research. Researchers and manufacturers extensively use the SPEC95 benchmark to study processor, memory system, and compiler performance. This benchmark is representative of computing-intensive applications. We ran the complete set of SPECint95 and SPECfp95 in our experiment, using the precompiled SPEC95 programs in the SimpleScalar tool set.

TPC benchmarks represent commercial workloads, which are widely used by computer manufacturers and database providers to test, evaluate, and demonstrate their products' performance. TPC-C is an online transaction processing benchmark. It is a mixture of read-only and update-intensive transactions that simulate a complex computing environment where a population of terminal operators execute transactions against a database.

Simulations and architecture parameters

The SimpleScalar tool set is a group of simulators for studying interactions between application programs and computer architecture. In particular, the sim-outorder simulator emulates typical ILP processors with the features of out-of-order execution, branch prediction, and nonblocking cache. It produces comprehensive statistics of the program execution. We have modified sim-outorder to include cached DRAM and other types of DRAM architecture simulations. In addition to the on-memory cache, we emulated the

Table 3. Architectural parameters of our simulation.

Structure	Parameter
CPU clock rate	500 MHz
L1 instruction cache	32 Kbytes, two-way, 32-byte block
L1 data cache	32 Kbytes, two-way, 32-byte block
L1 cache hit time	6 ns
L2 cache	2 Mbytes, two-way, 64-byte block
L2 cache hit time	24 ns
Memory bus width	32 bytes
Memory bus clock rate	83 MHz
On-memory cache block number	1 to 128
On-memory cache block size	2 to 8 Kbytes
On-memory cache associativity	One- to full-way
On-memory cache access time	12 ns
On-memory cache hit time	36 ns
On-memory cache miss time	84 ns
DRAM precharge time	36 ns
DRAM row access time	36 ns
DRAM column access time	24 ns

memory controller and a bus with contention. We also considered DRAM precharge, DRAM refresh, and processor/bus synchronization in the simulation.

We used sim-outorder to configure an eight-way processor, set the load/store queue size to 32, and set the register update unit size to 64 in the simulation. The processor allows up to eight outstanding memory requests, and the memory controller accepts up to eight concurrent memory requests. Table 3 gives other architectural parameters of our simulation. We used the processor and data bus parameters in a Compaq Workstation XP1000. The on-memory cache access time is the same as that in Hart's paper.[7] The on-memory cache hit time is the sum of the time for transferring the command/address to the cached DRAM (one bus cycle), the on-memory cache access time, and the time for the first chunk of data to be sent back (one bus cycle). The on-memory cache miss time is the sum of the time for transferring the command/address to the cached DRAM, the DRAM precharge time if the accessed memory bank needs precharge, the DRAM row access time, the time to transfer a row from the DRAM core to the on-memory cache (one bus cycle), the on-memory cache access time, and the time for the first chunk of data to be sent back.

The RDRAM connects to the processor by a 1-byte-wide, high-speed bus. The DRDRAM connects by a 1-byte-wide address bus and a 2-byte-wide data bus, and the bus speed is 400 MHz. Data is transferred on both edges of the block signal. For single-channel DRDRAM, the effective bandwidth is 1.6 Gbytes/s, which is not as large as the 2.6 Gbytes/s bandwidth of the bus used in our simulation. To make a fair comparison, we simulate the internal structure of the RDRAM and the DRDRAM, but set their bus simulation the same as other DRAMs. We will show that the advantage of cached DRAM is on its on-memory cache structure, not on its bus connection. In fact, the cached DRAM could also be connected to the processor by a high-speed, narrow bus.

Overall performance comparison

We measured the memory access portion of cycles per instruction (CPI) of the TPC-C workload and all the SPEC95 programs. To show the memory stall portion in each benchmark program, we used a method similar to the one that both Burger and Cuppu presented.[1-2] We simulated a system with an infinitely large L2 cache to eliminate all memory accesses. The application execution time on this system is called the base execution time. We also simulated a system with a perfect memory bus as wide as the L2 cache line, which separates the portion of the memory stall caused by the limited bandwidth. The CPI has three portions: the base or number of cycles spent for CPU operations and cache accesses; the latency or number of cycles spent accessing main memory; and the bandwidth or number of cycles lost due to the limited bus bandwidth. The memory access portion of the CPI is the sum of the latency and bandwidth portions.

We compared the cached DRAM with four DRAM architectures: SDRAM, ESDRAM, RDRAM, and DRDRAM. We used the TPC-C workload and eight SPECfp95 programs: tomcatv, swim, su2cor, hydro2d, mgrid, applu, turb3d, and wave5. We found that the memory access portions of the CPI of all SPECint95 programs and the two other SPECfp95 programs are very small. As a result, the programs' performances are not sensitive to the improvement of the main memory system. Although the memory access time reduction from using cached DRAM is

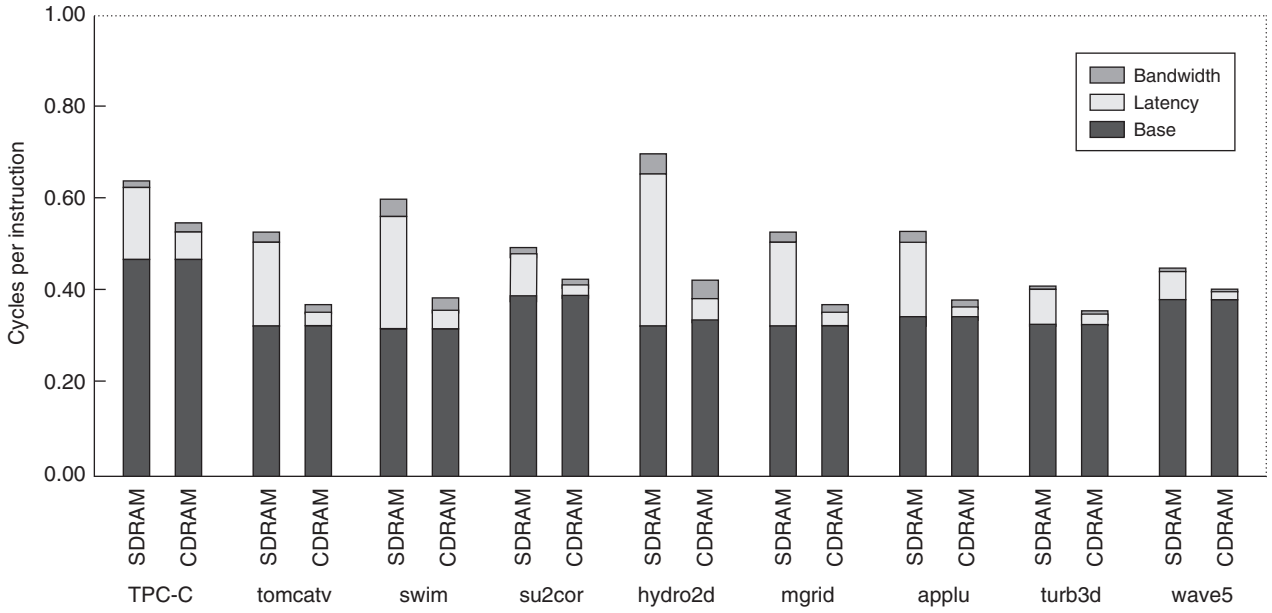

Figure 2. Cycles per instruction for the TPC-C workload and selected SPECfp95 programs.

also significant on those programs, the total execution time reductions are not significant.

On-memory cache organizations

We investigated the effects of changing the cache size and the cache associativity on the performance of the TPC-C workload and the eight selected SPECfp95 programs. Our experiments show that a small cache block size is not effective for the on-memory cache. The miss rates for TPC-C workload on a fully associative on-memory cache of 32 Kbytes with cache block sizes of 128 bytes, 256 bytes, and 512 bytes are 62, 36, and 22 percent. On the other hand, a small block number is also not effective for the on-memory cache. The miss rate for su2cor on a fully associative on-memory cache having four blocks of 4 Kbytes is as high as 80 percent. When the number of cache blocks increases to eight, the miss rate is still more than 40 percent. Only after the number of cache blocks increases to 16 is the miss rate effectively reduced to 5 percent. The experiments also show that the advantage of full associativity is significant. Direct-mapped, on-memory caches, even with many blocks, have high miss rates. This finding is important because most commercial DRAMs use the direct-mapped structure. Our study confirms the results reported in Koganti.[4] Because increasing the block size and number of blocks increases the on-memory cache's space requirement on the memory chip, there is a trade-off between performance and cost. A fully associative on-memory cache of 16×4 Kbytes is very effective for all workloads. The on-memory cache miss rates of the TPC-C workload and six of the SPECfp95 programs are below 5 percent, and the miss rates of the two other SPECfp95 programs are below 20 percent. Therefore, we used this on-memory cache configuration in the following experiments.

Cached DRAM and SDRAM

Figure 2 presents the CPIs and their decompositions for the TPC-C workload and the eight SPECfp95 programs on both the cached DRAM and the SDRAM. CPI reductions using the cached DRAM range from 10 to 39 percent. The effectiveness of the cached DRAM for reducing CPI is mainly determined by the percentage of memory access portion in each program's total CPI, which Table 4 (next page) lists. We show that CPI reduction by the cached DRAM mainly comes from reducing the latency portion of the CPIs; the bandwidth portion of CPIs is almost unchanged in each program.

Table 4. Memory access portion in CPI (cycles per instruction).

Program	Memory portion percentage
TPC-C	27
tomcatv	39
swim	47
su2cor	21
hydro2d	52
mgrid	37
applu	35
turb3d	20
wave5	15

Table 5. Reduction rate of the latency portion of cycles per instruction from using cached DRAM.

Program	Reduction rate percentage
TPC-C	62
tomcatv	84
swim	83
su2cor	75
hydro2d	83
mgrid	79
applu	87
turb3d	71
wave5	72

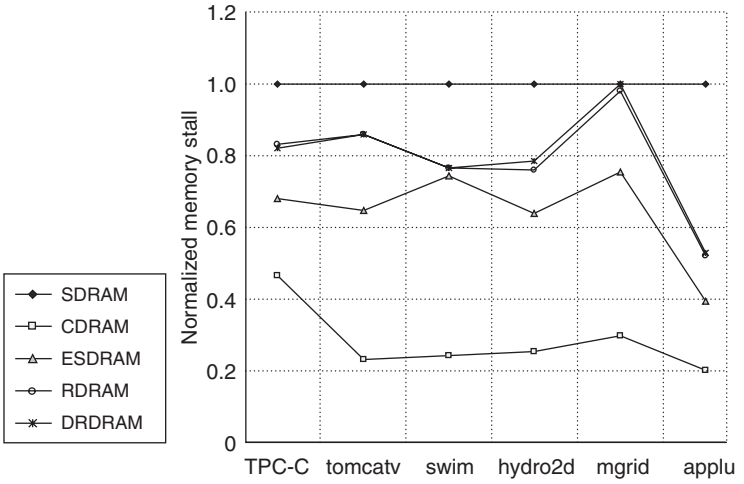

Figure 3. Comparison of memory stall times. The performance values are normalized to the memory access portion of cycles per instruction in the SDRAM configuration.

Table 5 presents reductions of CPI latency portions by using cached DRAM. For all selected SPECfp95 programs, cached DRAM reduces CPI latency portions by more than 71 percent. The reduction rate for the TPC-C workload is 62 percent. The program latency reduction rates are mainly determined by hit rates to the on-memory cache in the cached DRAM and to the row buffer in the SDRAM. For example, the on-memory cache hit rate of the cached DRAM for the TPC-C workload is 93 percent, while the row-buffer hit rate of the SDRAM is 50 percent. For tomcatv, the on-memory hit rate of the cached DRAM is 98 percent, whereas the row-buffer hit rate of the SDRAM is as low as 8 percent.

Cached DRAM and other DRAM architectures

Figure 3 shows the CPIs memory access portions of the TPC-C workload and five selected SPECfp95 programs on the cached DRAM and DRAM variants. Cached DRAM outperforms other DRAMs significantly on all programs. ESDRAM performs better than RDRAM and DRDRAM because of the low latency in accessing the on-memory cache. The cached DRAM outperforms the ESDRAM because the large number of blocks and the fully associative structure in the cached DRAM make the hit rate very high.

RDRAM and DRDRAM support high bandwidth by overlapping accesses among different banks. However, this is not very helpful in reducing access latency. Although both RDRAM and DRDRAM have many row buffers, the hit rates are still low because of the direct-mapped structure. In contrast, when the number of accesses to the DRAM core is very low, the cached DRAM acts almost as an SRAM main memory, providing both low latency and high bandwidth. As a result, the performance differences between the cached DRAM and the other DRAM architectures are large.

Our study shows that the performance of some programs on RDRAM or DRDRAM can differ significantly for the programs shown in Figure 3. The worst performance is for mgrid, and the best is applu. Both DRAMs perform better for programs that have many concurrent memory requests, because these DRAMs can effectively overlap this type of memory access. Figure 4 compares the distributions of concurrent memory requests by

mgrid and applu on the SDRAM when the memory system is busy. Our experiments show that the memory access concurrency degree of applu is much higher than that of mgrid. For example, 27 percent of memory accesses in applu have a concurrency degree of 8, and the percentages of the memory accesses with concurrency degrees of 3 to 7 range from 5 to 13 percent. In contrast, the majority of memory accesses in mgrid have concurrency degrees of 1 (48 percent of the total memory accesses) or 2 (29 percent of the total memory accesses). This explains why RDRAM and the DRDRAM are more effective for applu than mgrid, although both programs have comparable memory access portions in their CPIs, as shown in Table 4.

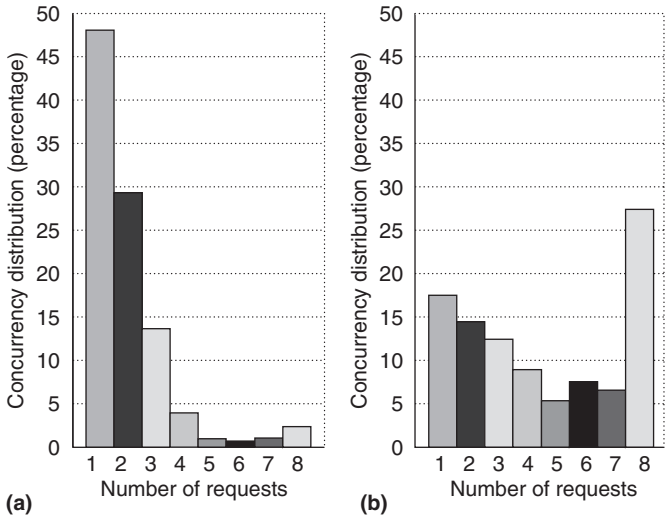

Figure 4. Distribution of the number of concurrent memory requests on the SDRAM system when the memory is busy: The mgrid (a) program has a much lower concurrency than that of applu (b).

Increasing the ILP degree

Commercial computer architectures extensively use wide-issue processors. When the ILP degree increases, the processor increases demand on the main memory system. Thus, it is important to understand how cached DRAM performs as ILP degrees change. We compared the performance of cached DRAM and SDRAM with four-, eight-, and 16-way-issue processors.

Figure 5 shows CPI as the ILP degree changes. The CPI's base portion decreases proportionally for all programs as the ILP degree increases. This means that, with an ideal main memory system, the performance always improves as the ILP degree increases.

Our experiments show that the CPIs of three programs on the 16-way issue processor with the SDRAM are slightly higher than those of the eight-way issue processor with the SDRAM. The performance degradation mainly comes from the heavy demand to the main memory system. This heavy demand causes congestion at the main memory system, and increases the

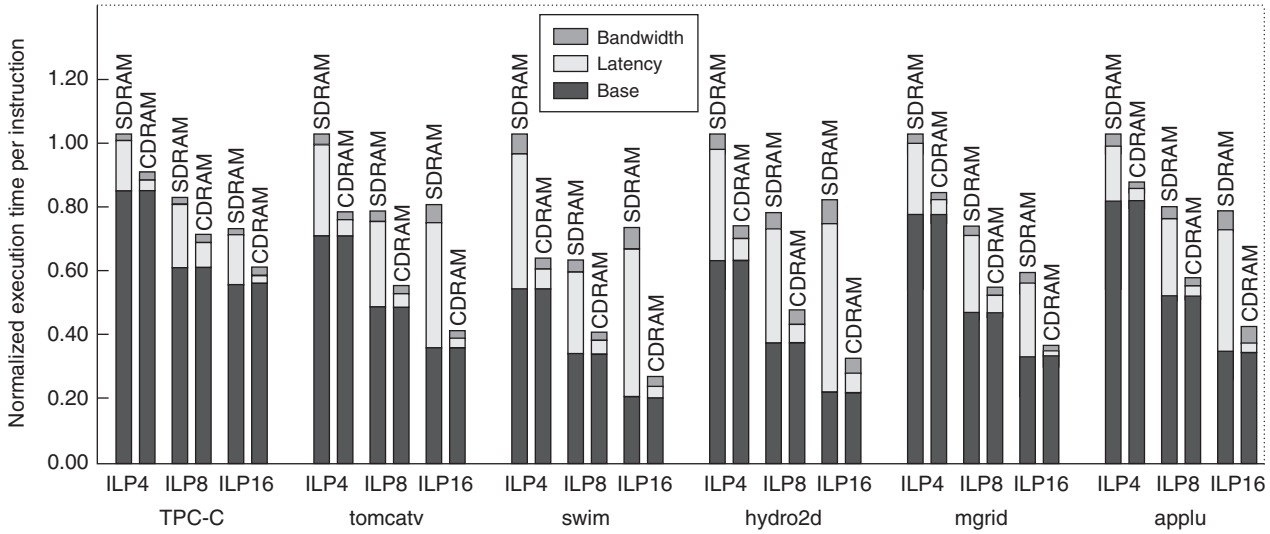

Figure 5. Cycles per instruction and their decompositions for the TPC-C workload and five SPECfp95 programs as the ILP degree changes.

memory access latency due to queuing effects. Consequently, the rate of instructions retiring from the instruction window decreases, reducing the speed of new instructions entering the instruction window. For example, the instruction dispatch rate on the 16-way issue processor is 30 percent less than that of the eight-way issue processor for hydro2d. The average time that an instruction stays in the instruction window is 43 processor cycles for the eight-way issue processor, but increases to 88 processor cycles for the 16-way issue processor.

In contrast, the cached DRAM performs consistently well as the ILP degree increases. Because the on-memory cache hit rate is high, the cached DRAM can support high-bandwidth and low-latency accesses. Thus, congestion at the memory system is not as severe as on SDRAM. The memory stall time does not increase as the ILP degree increases from 8 to 16.

The processor's inability to effectively tolerating long memory access latency caused by SDRAM will eventually offset the benefit of reducing computing time due to the ILP degree increase. In contrast, cached DRAM effectiveness will increase as the ILP increases to a much higher degree.

Exploiting row buffer locality

Researchers have made efforts to exploit the locality in row buffers to reduce DRAM access latency and improve DRAM bandwidth utilization. For example, most contemporary DRAMs support both the open and close page modes. In the open page mode, data in the row buffer is kept valid after the current access finishes, and only the column access is necessary for the next access as long as the data is in the row buffer (a row buffer hit). The effectiveness of open page mode depends mainly on the hit rate to the row buffers. The structure of the row buffer is comparable to that of a direct-mapped cached DRAM.

Contemporary DRAM chips support increasingly more memory banks and row buffers. However, the row buffer hit rate is still low. Studies have shown that the low hit rate is directly related to the memory-interleaving scheme—that is, how physical memory addresses are mapped onto DRAM banks.[10-11] To exploit row buffer locality, the memory space is usually interleaved page by page; this is known as page interleaving. All conventional interleaving schemes use an address bit portion as the bank index to determine to which bank the address is mapped. The bank index used by the page-interleaving scheme is a portion of the address bits used for cache set index. Because of this connection between memory accesses and cache accesses, cache conflict misses will result in row buffer conflicts, and write backs will compete with current reads for the same row buffer.[10]

A permutation-based interleaving scheme uses two portions of address bits to generate new bank indices. One portion is the bank index used in the conventional page-interleaving scheme, and the other portion comes from the address bits used for cache tags.[11-12] Using these two portions as inputs, an XOR operator outputs a new bank index for an address mapping. It is still a page-interleaving scheme, but the portion from tags permutes the mapping of pages to banks. Consequently, accesses causing row buffer conflicts in the conventional page-interleaving scheme are distributed to different banks without changing the locality in the row buffer. The results show that the scheme can significantly improve the row buffer hit rate.[10-11]

Because cached DRAM and the permutation-based interleaving scheme share the same objective of reducing conflicts in the on-memory cache or row buffers, it is necessary to compare their performance potential. Cached DRAM usually uses full or high associativity. The permutation-based interleaving scheme uses a special mapping to reduce row buffer conflicts without changing the direct-mapped structure. The cached DRAM approach has several advantages over the permutation-based interleaving scheme. Accesses to the on-memory cache are faster than accesses to the row buffer. Due to its high associativity, the on-memory cache hit rate is higher than the row buffer hit rate under the permutation-based scheme. The on-memory cache can be accessed independently with the DRAM core. Even if a DRAM bank is in precharge or in a row access, its cached data in the SRAM can be accessed simultaneously. In contrast, the data in the row buffer is lost as soon as the precharge starts. Finally, the cached DRAM organization allows more on-memory cache blocks than the number of banks, which is beneficial to

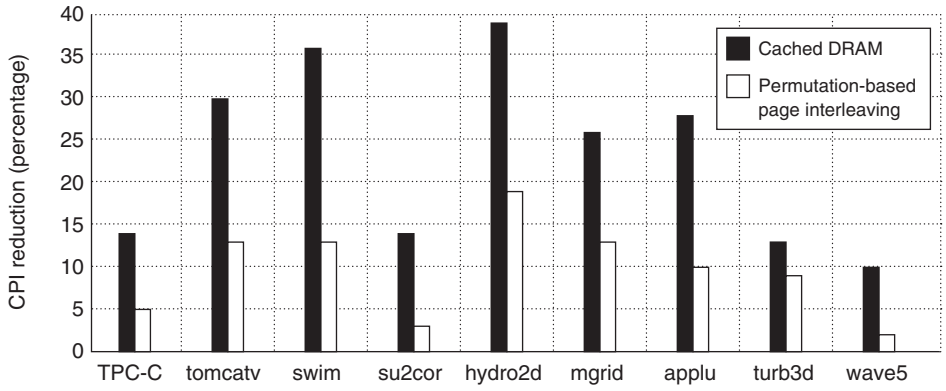

Figure 6. Cycles per instruction reduction for cached DRAM and XOR-based interleaving scheme.

DRAM systems with a limited number of banks. However, the permutation-based interleaving scheme requires little additional cost and does not require any change in the DRAM chip. In contrast, cached DRAM requires an additional chip area for the SRAM cache and additional circuits for cache management.

Figure 6 compares the performance improvements between cached DRAM and the permutation-based interleaving scheme over SDRAM systems, using the identical simulation configuration as Table 3 with the following differences: The cached DRAM has 16×4 Kbytes of on-memory cache, whereas the permutation-based interleaving scheme is used for 32 banks and 32- $\times$ 2-Kbyte row buffers. We show that cached DRAM reduces the CPI by 23 percent on average, while the average CPI reduction by the permutation-based scheme is 10 percent.

Our study provides three new findings: cached DRAM consistently improves performance as the ILP degree increases; contemporary DRAM schemes do not exploit memory access locality as effectively as cached DRAM; and compared with a highly effective permutation-based DRAM interleaving technique, cached DRAM still substantially improves performance.

Acknowledgments

We thank Jean-Loup Baer and Wayne Wong at the University of Washington for reading a preliminary version of this article and for their insightful and constructive comments and discussions. We also thank our colleague Stefan A. Kubricht for reading this article and providing comments and corrections. Finally, we thank the anonymous referees for their constructive comments on our work. This work is supported in part by the National Science Foundation under grants CCR-9812187, EIA-9977030, and CCR-0098055; the Air Force Office of Scientific Research under grant AFOSR-95-1-0215; and Sun Microsystems under grant EDUE-NAFO-980405.

References

1. D. Burger, J.R. Goodman, and A. Kägi, "Memory Bandwidth Limitations of Future Microprocessors," *Proc. 23rd Ann. Int'l Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., 1996, pp. 78-89.
2. V. Cuppu et al., "A Performance Comparison of Contemporary DRAM Architectures," *Proc. 26th Ann. Int'l Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. 222-233.
3. W.-C. Hsu and J.E. Smith, "Performance of Cached DRAM Organizations in Vector Supercomputers," *Proc. 20th Ann. Int'l Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., 1993, pp. 327-336.
4. R.P. Koganti, and G. Kedem, *WCDRAM: A Fully Associative Integrated Cached-DRAM with Wide Cache Lines*, tech. report CS-1997-03, Dept. of Computer Science, Duke Univ., Durham, N.C., 1997.
5. W. Wong and J.-L. Baer, *DRAM On-Chip Caching*, tech. report UW CSE 97-03-04,

Dept. of Computer Science and Engineering, Univ. of Washington, 1997.
6. H. Hidaka et al., "The Cache DRAM Architecture: A DRAM with an On-Chip Cache Memory," *IEEE Micro*, vol. 10, no. 2, Mar. 1990, pp. 14-25.
7. C.A. Hart, "CDRAM in a Unified Memory Architecture," *Proc. 39th Int'l Computer Conf.* (COMPCON 94), IEEE CS Press, Los Alamitos, Calif., 1994, pp. 261-266.
8. Y. Katayama, "Trends in Semiconductor Memories," *IEEE Micro*, Vol. 17, No. 6, Nov./Dec. 1997, pp. 10-17.
9. D.C. Burger and T.M. Austin, *The SimpleScalar Tool Set, Version 2.0*, tech. report CS-TR-1997-1342, Dept. of Computer Sciences, Univ. of Wisconsin, Madison, 1997.
10. Z. Zhang, Z. Zhu, and X. Zhang, "A Permutation-Based Page Interleaving Scheme to Reduce Row-Buffer Conflicts and Exploit Data Locality," *Proc. 33rd Ann. IEEE/ACM Int'l Symp. Microarchitecture* (MICRO-33), IEEE CS Press, Los Alamitos, Calif., 2000, pp. 32-41.
11. W.-F. Lin, S. Reinhardt, and D.C. Burger, "Reducing DRAM Latencies with an Integrated Memory Hierarchy Design," *Proc. 7th Int'l Symp. High-Performance Computer Architecture* (HPCA-7), IEEE CS Press, Los Alamitos, Calif., 2001, pp. 301-312.

Zhao Zhang is a PhD candidate in computer science at the College of William and Mary. His research interests include computer architecture and parallel systems. He has a BS and MS in computer science from Huazhong University of Science and Technology, China. He is a student member of the IEEE and the ACM.

Zhichun Zhu is a PhD candidate in computer science at the College of William and Mary. Her research interests include computer architecture and computer system performance evaluation. She has a BS in computer science from Huazhong University of Science and Technology, China. She is a student member of the IEEE and the ACM.

Xiaodong Zhang is a professor of computer science at the College of William and Mary. He is also the program director of the Advanced Computational Research Program at the National Science Foundation, Washington, D.C. His research interests include parallel and distributed systems, computer system performance evaluation, computer architecture, and scientific computing. He has a BS in electrical engineering from Beijing Polytechnic University, China, and an MS and PhD in computer science from the University of Colorado at Boulder. He is an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* and has chaired the IEEE Computer Society Technical Committee on Supercomputing Applications. He is a senior member of the IEEE.

Direct questions and comments about this article to Xiaodong Zhang, Dept. of Computer Science, College of William and Mary, Williamsburg, VA 23187-8795; zhang@cs.wm.edu.