Data Parallel Address Architecture

Jung Ho Ahn and William J. Dally
Computer Systems Laboratory
Stanford University, Stanford, California 94305, USA
{gajh,billd}@cva.stanford.edu

Abstract— **Data parallel memory systems must maintain a large number of outstanding memory references to fully use increasing DRAM bandwidth in the presence of increasing latency. At the same time, the throughput of modern DRAMs is very sensitive to access patterns due to the time required to precharge and activate banks and to switch between read and write access.**

To achieve memory reference parallelism a system may simultaneously issue references from multiple reference *threads*. Alternatively multiple references from a single thread can be issued in parallel. In this paper we examine this tradeoff and show that allowing only a single thread to access DRAM at any given time significantly improves performance by increasing the locality of the reference stream and hence reducing precharge/activate operations and read/write turnaround. Simulations of scientific and multimedia applications show that generating multiple references from a single thread gives, on average, 17% better performance than generating references from two parallel threads.

I. Introduction

Processor performance has been increasing more rapidly than memory performance making memory bandwidth and latency the bottlenecks in many applications. The performance of modern DRAMs is very sensitive to access patterns due to their organization as multiple banks where each bank is a 2-dimensional memory array. Memory systems consisting of several address-interleaved *channels* of DRAMs are even more sensitive to access patterns. To achieve high performance an access pattern must have sufficient parallelism to tolerate memory latency while keeping the memory bandwidth of all the banks and channels occupied. The access pattern must also exhibit locality to minimize activate/precharge cycles, avoid bank conflicts, and avoid read/write turnaround penalties.

Multimedia and scientific applications have access patterns that contain multiple *streams* or *threads* of accesses. For example, in a vector or stream processor each vector or stream load or store is a thread of related memory references. Also, in a DSP with a software managed local memory, each DMA transfer to or from local memory can be thought of as a thread of memory references.

A data parallel memory system can exploit parallelism both within a thread — by generating several thread accesses per cycle — and/or across threads — by generating accesses from different threads in parallel. Exploiting parallelism across threads may result in frequent read/write turnaround and numerous precharge and activate cycles as there is little locality between two threads. Memory access scheduling [7] addresses this performance degradation by reordering memory references to enhance locality.

Manuscript submitted: 24 June 2005. Manuscript accepted: 12 Oct. 2005. Final manuscript received: 19 Oct. 2005.

TABLE I
DRAM TIMING PARAMETERS

	SDRAM	DDR2	GDDR3	XDR
Pin BW (Mbps)	133	667	1,600	4,000
tCK (ns)	7.5	3	1.25	2
tRC (tCK)	9	17	35	24
tDWR [tDRW] (tCK)	1 [4]	8 [4]	14 [9]	10 [9]
tWRBUB [tRWBUB] (tCK)	0 [1]	0 [1]	2 [2]	3 [3]
Minimum burst (tCK)	1	2	2	2

In this paper we investigate an alternative and complementary method of improving memory locality in data parallel systems: using only a single, wide address generator (AG). With this approach, all accesses from one thread are sent to the memory system before any accesses from another thread are generated. This enhances locality — by eliminating interference between threads — at the possible expense of load balance — if the single thread does not access the memory channels (MCs) and banks evenly. This single thread approach can be used with memory access scheduling to further enhance locality. Compared to an approach using two narrower AGs, the single AG configuration gives an average 17% performance improvement in memory trace execution on a set of multimedia and scientific applications.

The rest of this paper is organized as follows. Section II summarizes the characteristics of modern DRAM architecture, Section III details micro-architecture of data parallel memory system, Section IV provides experimental setup, Section V analyzes the microbenchmark and application memory trace results, and Section VI concludes the paper.

II. DRAM Architecture

Modern DRAMs such as SDR, DDR, DDR2, GDDR3, RDRAM, and XDR [4] consist of a number of memory banks where each bank is a two-dimensional array. SDR and RDRAM parts are described in detail in [2]. A DRAM address is specified by bank, row, and column numbers. Row and column addresses are delivered to the DRAM through the same request path, and both read and write data pass through a bi-directional data path. All the banks within a DRAM share

Posted to IEEE & CSDL on 1/30/2006
DOI 10.1109/L-CA.2006.4 1556-6056/05/$20.00 © 2005 Published by the IEEE Computer Society

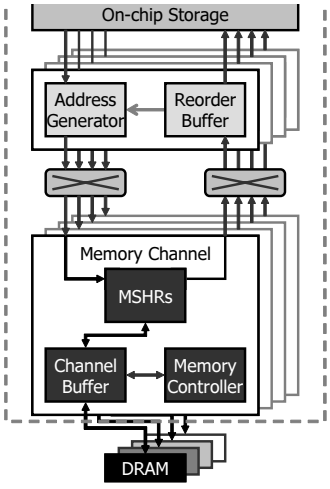

Fig. 1. Streaming memory system

the request and data paths. Accessing data in each bank must follow certain rules and timing constraints.

In order to access a location, the entire row containing that location must first be activated. Any location within that row can then be accessed via its column address. If the next location to be accessed is in a different row of the same bank (an internal bank conflict), the old row must be precharged and the new row activated before the new location can be accessed via its column address. Hence this second access requires two row level commands (precharge and activate) followed by one column level command (read or write).

Table I summarizes key timing parameters of modern DRAMs. Data pin bandwidth has been increasing rapidly while command bandwidth specified by `tCK` has been increasing more slowly. This results in increasing access granularity. Also, the time between successive row activations to the same bank (`tRC`) has been growing rapidly, making internal bank conflicts more costly. A write command and data followed by a read must be separated by `tDWR` and `tWRBUB` cycles (`tDRW` and `tRWBUB` cycles for a read followed by a write). This bus turnaround time has also been increasing over time making DRAM performance sensitive to varying access type. Taken together, these trends show that DRAM performance is becoming more sensitive to the access pattern applied.

III. Data Parallel Memory Systems

Figure 1 shows a typical data parallel memory system that moves data between on-chip storage (SRF [1], vector registers [8], or local store [5]) and one or more channels of off-chip DRAM. One or more AGs generate streams of accesses that are routed to one or more MCs via a crosspoint switch that selects the MC based on the low bits of the memory address. The MC performs the requested access and, for a read, sends the data to the reorder buffer (RB) associated with the originating AG. The RBs collect out-of-order replies from the MCs and return the collected data to the on-chip storage in order.

Within each MC, miss status holding registers (MSHRs) [6] act as a small non-blocking cache, keeping track of in-flight references, and performing read and write coalescing of accesses. The width of MSHR entries matches DRAM burst length. Accesses that cannot be handled in the MSHRs are forwarded to the channel buffer. The memory controller performs memory access scheduling [7] on the pending accesses in the channel buffer, selecting one access on each DRAM command cycle.

To keep multiple MCs busy, a data parallel memory system must generate several accesses per cycle. This can be accomplished by having multiple AGs, by having each AG generate multiple accesses per cycle, or both. Using a single, wide AG enhances locality by eliminating interference between memory threads. It also simplifies hardware design. On the other hand, using multiple AGs results in better load balance when the accesses from multiple memory threads are more evenly distributed over channels than those from a single memory thread, alleviating the *hot memory channel* effect.

IV. Experimental setup

We compare the performance of single-wide and multiple-narrow AG configurations on memory traces of microbenchmarks and of several multimedia and scientific applications. Each trace is run on a simulated single-node stream processor.

A. Test Applications

Table II summarizes the test programs used for evaluation. Microbenchmarks listed in the upper half of the table stress particular aspects of memory system performance, e.g., DRAM read/write turnaround penalty (benchmarks named with rw) and internal bank conflict (benchmarks with cf). In the table, record length refers to the length (in words) of a record that is transferred as a contiguous series of words and stride refers to the difference in word address between the first words of two successive records. Eight access threads are included in each microbenchmark, and each thread accesses $4,096$ words except in the 48x48rw benchmark, which accesses $3,840$ words per thread.

Multimedia and scientific applications listed in the lower half of the table were modified from their original implementation [1] [3] to enable them to run on the simulated configurations.

B. Stream Processor Architecture

Simulations were performed using a cycle-accurate simulator for a single node of the Imagine stream processor [1]. Imagine uses a software managed *stream register file* (SRF) to stage data to and from external DRAM memory. The simulated Imagine operates at 1 GHz, the DRAM burst length (`1B`) is 4 words (2 `tCK` cycles), each DRAM row contains 1K words, each DRAM chip contains 8 banks, and XDR DRAM parameters (see Table I) are used unless mentioned otherwise.

For evaluation, the number of AGs is varied from 1 to 4 while holding total AG bandwidth constant. That is we

TABLE II
BENCHMARK PROGRAMS

Name	Description
1x1rd	record length = stride = 1 and read only. unit stride read request threads with parallel threads to different rows in different DRAM banks
1x1rw	same as 1x1rd except that read and write request threads are interleaved
1x1rdcf	same as 1x1rd except with concurrent threads to different rows in the same DRAM bank
1x1rwcf	same as 1x1rdcf except that read and write request threads are interleaved
48x48rw	same as 1x1rw except record length = stride = 48
cr1rd	constrained random, record length = 1 and read only. random access read threads to a 64K word range
r1rd	same as cr1rd except 4M word index range
r4rw	random, record length = 4. random access read and write threads to a 4M word range
DEPTH	stereo depth extraction between two images from different horizontal angles
MPEG	encoding 3 frames of 360x288 video images according to the MPEG-2 standard
RTSL	rendering the SPECviewperf benchmark using the Stanford Real-Time Shading language
QRD	converting a complex matrix into an upper triangular and orthogonal matrix
FEM	finite element code solving systems of first-order conservation laws on unstructured meshes
MOLE	a molecular dynamics solver based on solving Newton's equations of motion

compare 1 AG with 8 requests per cycle, 2 AGs with 4 requests per cycle, and 4 AGs with 2 requests per cycle. The simulated configuration uses 4 MCs (nMC) unless otherwise specified. Each MC processes a maximum of one word per cycle. The simulated memory access scheduler uses either an open column (opcol) or an inorder look-ahead (inorderla) scheduling policy. Opcol policy precharges a row when there is no pending access for that row and at least one pending access to another row in the same bank. If multiple DRAM banks are ready to accept commands, it gives higher priority to column commands and picks older accesses when all else is equal. Other scheduling policies allowing access reordering with priority provide similar performance as shown in [7]. Inorderla policy issues the column commands in the order requests arrive. It looks ahead to issue row commands when possible and also gives priority to older accesses.

V. RESULTS

Figure 2 shows the relative execution time of microbenchmarks and applications on six simulated configurations. The configurations include 1, 2, and 4 AGs and both opcol and inorderla scheduling policies. The runtime of each configuration is normalized to that of opcol using a single AG.

In Figure 2(a), runtime of 1x1rd is constant across the configurations. This is expected because there is neither internal bank conflict nor read/write turnaround penalty experienced with this benchmark. For 1x1rw using more than one AG increases runtime due to DRAM read/write turnaround penalty experienced when interleaving memory access threads. Internal bank conflict causes performance to degrade with multiple AGs on the 1x1rdcf benchmark. Both bank conflict and read/write turnaround affect performance for the 1x1rwcf benchmark. The degradation due to both read/write turnaround and internal bank conflicts are more pronounced with the inorderla policy because the opcol policy is able to restore some of the locality lost by interleaving.

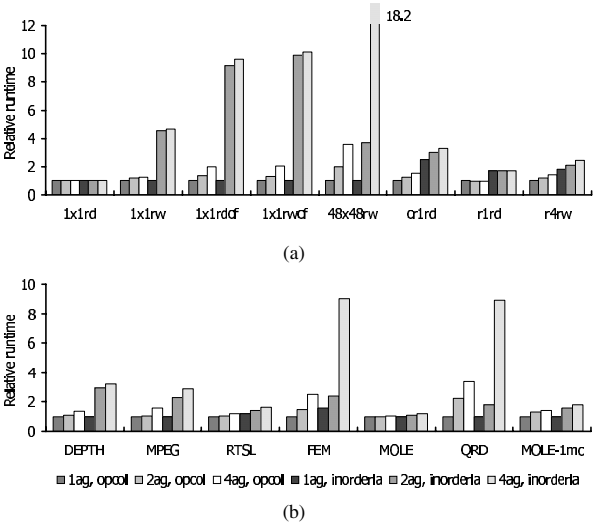

Fig. 2. Relative runtime of (a) synthetic programs, (b) applications

The execution time of the 48x48rw benchmark is affected by a transient load imbalance due to the clustered generation of addresses. The AG first fetches the first word of the first $nC = 8$ records, one per cluster, then the second word of each of these records, and so on. These requests are sorted by address so that in this case, where stride is a multiple of $nMC \cdot nC$, the first $lB \cdot nC$ words all go to the same MC. Multiple AG configurations make the situation even worse by interleaving read and write threads causing further degradation due to read/write turnaround.

For the random access cases, opcol policy gives better performance than inorderla policy since reordering read and write commands brings more flexibility in utilizing available DRAM command slots and finding the requests whose addresses belong to the same row. In r1rd, execution time doesn't depend on the number of AGs since there is no difference in spatial locality coming from different AGs. In group cr1rd, multiple AG configurations have lower performance because interleaving reduces locality — in effect increasing the range over which addresses are distributed.

Table III summarizes the memory access characteristics of the scientific and media applications and Figure 2(b) shows the relative runtime of the memory traces taken from these applications. The data in Table III, especially average record

TABLE III
APPLICATION CHARACTERISTICS

Appli-cation	Avg strided (W) record length	Avg strided (W) stream length	Avg indexed (W) record length	Avg indexed (W) stream length	Strided access	Read access
MOLE	1	480	9	3252	9.9%	99.5%
MPEG	1	1515	1	1280	90.1%	70.2%
QRD	115	1053	N/A	N/A	100%	69.0%
DEPTH	1.96	1802	1	1107	46.6%	63.0%
FEM	12.4	1896	24	3853	48.8%	74.0%
RTSL	4	1170	1	264	65.1%	83.5%

length and fraction of strided and read accesses, can be used to explain the runtime trends of the applications. In MOLE, gathers of 9-word records dominate the stream memory accesses. Since the range of indices is small, performance is similar to the cr1rd microbenchmark. The large record length of the QRD and FEM applications make their trends similar to 48x48rw. MPEG and DEPTH have performance similar to the strided microbenchmarks because they perform mostly read accesses with small record length and their indexed accesses are distributed over a small range. Two thirds of memory accesses in RTSL are strided with 4-word records and the other third are indexed with a large range. As a result, the performance is similar to a blend of 1x1rw and r1rd.

The runtime of MOLE is slightly longer with a single AG due to a slight load imbalance between MCs. The fact that multiple AG configurations take more time at 1-MC cases supports this analysis. With a single MC (MOLE-1mc) the load imbalance no longer exists and the single AG gives the best performance.

Figure 3 shows the sensitivity of memory system bandwidth to write-to-read turnaround time and DRAM burst length for 1 and 2 AGs and a single MC using the `opcol` scheduling policy. In Figure 3(a), `tDWR` is set to 0.5x, 1x, 2x, and 4x its baseline value. For benchmarks that involve both read and write operations, longer turnaround time reduces bandwidth for the 2-AG cases. The 1-AG cases are largely insensitive to turnaround time.

Figure 3(b) shows memory system bandwidth of the microbenchmarks with DRAM burst lengths of 1, 2, 4 and 8 `tCK` duration. The performance of strided access benchmarks on 2-AG cases improves as the DRAM burst gets longer because the two address threads are interleaved with a coarser granularity — resulting in less interference. At the same time, increasing DRAM burst length hurts the performance of indexed accesses because less of the data accessed in a long burst is likely to be needed.

VI. CONCLUSION

We have shown that using a single wide address generator rather than multiple narrow address generators improves data-parallel memory system performance. The bandwidth of modern DRAM memory systems is very sensitive to access

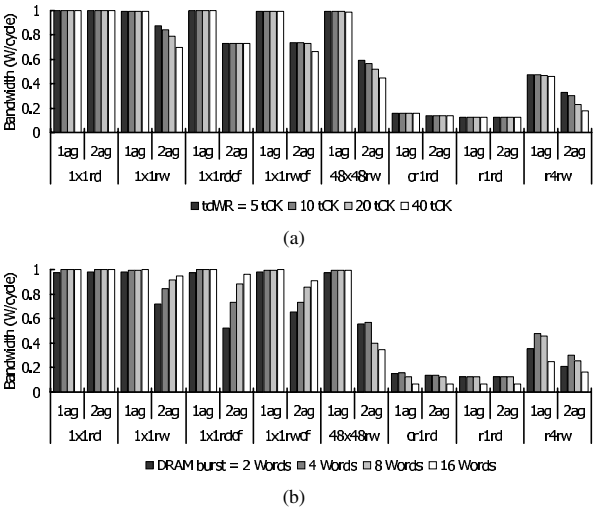

Fig. 3. Runtime of synthetic programs varying (a) DRAM `tDWR`, (b) DRAM burst length - 1 MC, `opcol` policy configuration

patterns — read/write turnaround, internal bank conflicts, and limited active-row reuse all reduce performance. Moreover this degradation in DRAM performance due to unfavorable access patterns is growing over time as DRAM access granularity and read/write turnaround times increase. Using a single wide address generator reduces this degradation by eliminating the interference that results when multiple address *threads* are interleaved. The load imbalance over memory channels caused by processing a single address thread at a time was negligible on the benchmarks we analyzed. The use of a single wide address generator to improve memory system performance is complementary to, and can be used in conjunction with, memory access scheduling which reorders memory requests to improve spatial locality and hence DRAM performance.

REFERENCES

[1] J. H. Ahn, W. J. Dally, B. Khailany, U. J. Kapasi, and A. Das, "Evaluating the Imagine Stream Architecture," in *Proceedings of the 31st Annual International Symposium on Computer Architecture*, Jun 2004, pp. 14–25.

[2] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, "A performance comparison of contemporary DRAM architectures," in *Proceedings of the 26th Annual International Symposium on Computer Architecture*, May 1999, pp. 222–233.

[3] W. J. Dally, P. Hanrahan, M. Erez, T. J. Knight, F. Labonte, J. H. Ahn, N. Jayasena, U. J. Kapasi, A. Das, J. Gummaraju, and I. Buck, "Merrimac: Supercomputing with streams," in *SC'03*, Nov 2003.

[4] ELPIDA Memory, Inc, "512M bits XDR^{TM} DRAM," http://www.elpida.com/pdfs/E0643E20.pdf.

[5] B. Flachs *et al.*, "A Streaming Processing Unit for a Cell Processor," in *IEEE International Solid-State Circuits Conference*, Feb 2005, pp. 134–135.

[6] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," in *Proceedings of the 8th Annual Symposium on Computer Architecture*, May 1981, pp. 81–87.

[7] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory Access Scheduling," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, Jun 2000, pp. 128–138.

[8] R. M. Russell, "The CRAY-1 computer system," *Communications of the ACM*, vol. 21, no. 1, pp. 63–72, Jan 1978.