The Design Space of Data-Parallel Memory Systems

Jung Ho Ahn, Mattan Erez, and William J. Dally
Computer Systems Laboratory
Stanford University, Stanford, California 94305, USA
{gajh,merez,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 rising latencies. Additionally, throughput is increasingly sensitive to the reference patterns due to the rising latency of issuing DRAM commands, switching between reads and writes, and precharging/activating internal DRAM banks. We study the design space of data-parallel memory systems in light of these trends of increasing concurrency, latency, and sensitivity to access patterns. We perform a detailed performance analysis of scientific and multimedia applications and micro-benchmarks, varying DRAM parameters and the memory-system configuration. We identify the interference between concurrent read and write memory-access threads, and bank conflicts, both within a single thread and across multiple threads, as the most critical factors affecting performance. We then develop hardware techniques to minimize throughput degradation. We advocate either relying on multiple concurrent accesses from a single memory-reference thread only, while sacrificing load-balance, or introducing new hardware to maintain both locality of reference and load-balance between multiple DRAM channels with multiple threads. We show that a low-cost configuration with only 16 *channel-buffer entries achieves over* 80% *of peak throughput in most cases.*

1. Introduction

The performance demands of multimedia and scientific applications continue to rise rapidly. While data-parallel processors have been scaling to keep pace with the computation demands, memory system performance has increased at a slower rate and hence has become a bottleneck. In this paper we focus on the performance trends of DRAM and their effect on memory system architecture.

The demand for aggregate memory bandwidth is met by building memory systems using multiple address-interleaved *memory channels* and implementing each channel using high-bandwidth DRAM components. Such *data-parallel memory systems* (DPMSs), however, are very sensitive to access patterns. This sensitivity is compounded by the fact that the latency of modern DRAMs has been improving very slowly. Therefore, to achieve high performance on a data-parallel memory system, an access pattern must have sufficient parallelism to saturate the memory bandwidth while tolerating memory latency – i.e., the number of outstanding memory references must equal at least the product of bandwidth and latency. Additionally, to reduce sensitivity and maintain near-peak throughput, the access pattern must also exhibit locality to minimize activate/precharge cycles, avoid bank conflicts, and minimize read-write turnaround penalties.

To achieve the required levels of concurrency, DPMS designs have turned to the inherent parallelism of memory references common to multimedia and scientific applications. A characteristic of these application domains is multiple *streams* or *threads*, each containing a large number of memory accesses. For example, in a vector [1] or stream [2] 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 is a *thread* of memory references. A DPMS exploits this parallelism by generating multiple references per cycle within a thread, by generating concurrent accesses across threads, or both. Accesses from the same thread tend to display locality, yet this locality may lead to load imbalance between the different channels and lower the effective bandwidth. Load balancing is achieved by interleaving accesses from multiple threads at the expense of locality and potentially reduced performance due to access-pattern sensitivity. Based on current technology trends, our experiments show that this loss of locality is particularly problematic when bank conflicts occur within a single DRAM, and when threads performing reads and writes are interleaved. The reason is the long delays associated with switching between read and write accesses, and between rows within a single bank, on a modern DRAM.

A second memory-system architecture trend is to rely on long cache lines and long DRAM bursts. Long DRAM

This work was supported, in part, by the Department of Energy ASC Alliances Program, Contract Lawrence Livermore National Laboratory B523583, with Stanford University.

SC2006 November 2006, Tampa, Florida, USA
0-7695-2700-0/06 $20.00 ©IEEE

bursts are used to maintain high bandwidth from a single DRAM in face of rapidly increasing access latencies. Since DRAM commands must be separated by several cycles, long bursts are used to utilize internal DRAM bandwidth. However, this large granularity introduced by the memory system does not always match application access granularity. For example, if an application performs random references, only a fraction of each long burst contains useful data and effective performance is significantly reduced.

In this paper, we explore, in detail, the design space of data-parallel memory systems [3] in light of these new trends in modern DRAM architectures and application characteristics. We experiment with several scientific and multimedia benchmarks as well as micro-benchmarks, and show that the most critical performance factors are high read-write turnaround penalties and internal DRAM bank conflicts. A second important aspect is the detrimental effect of increasing burst lengths on applications that display random-access memory reference patterns. To better characterize this trend, we develop an accurate analytical model for the effective random-access bandwidth given DRAM technology parameters and the burst-length. A sensitivity study of the amount of hardware buffering shows that a reasonable 16-entry channel-buffer achieves near-maximal performance. We also experiment with varying DRAM timing and measure a significant bandwidth drop as latencies are increased when multiple threads are accessed concurrently and when the applications display random-access patterns.

Based on our experimental results, we develop techniques for improving the effective bandwidth of modern DPMSs. Our designs are based on the observation that locality and interference are the most critical factors affecting modern DRAM performance. We investigate using a single, wide address generator, which sacrifices load-balancing in favor of locality, and suggest a novel *channel-split* architecture that regains load-balance through additional hardware.

The main contributions of the paper are summarized below:

- We establish that the sensitivity of DRAM throughput to application access patterns is increasing rapidly as a result of the growing ratio between DRAM bandwidth and latency.
- We detail the design space of DPMSs, and explore performance behavior in light of DRAM technology trends.
- We show the importance of maintaining locality due to the high access-pattern sensitivity.
- We suggest a DPMS design that focuses on locality, and introduce the novel channel-split DPMS configuration that achieves locality without sacrificing channel load balance.
- We identify the trends of large granularity DRAM bursts, and long-latency DRAM commands, as a potential problem affecting the throughput of applications that perform random accesses, and develop an accurate analytical model for the expected effective bandwidth.

	SDRAM	DDR2	GDDR3	XDR
Pin bandwidth (Mbps)	133	667	1600	4000
tCK (ns)	7.5	3	1.25	2
$tRCDW\backslash tRCDR$ (tCK)	3\3	4\4	8\12	3\7
$tCWD\backslash tCAC$ (tCK)	0\3	3\4	6\11	3\7
tRC (tCK)	9	17	35	20
tRR (tCK)	2	3	8	4
$tDWR\backslash tDRW$ (tCK)	1\4	8\4	14\9	10\9
$tWRBUB\backslash tRWBUB$ (tCK)	0\1	0\1	2\2	3\3
Minimum burst (tCK)	1	2	2	2

Table 1: DRAM timing parameters based on [4]–[7]

The rest of the paper is organized as follows: Section 2 summarizes the characteristics of modern DRAM architectures; Section 3 explores the design-space of data parallel memory systems; Section 4 details the experimental setup; Section 5 presents results for our application benchmarks; Sections 6 and 7 discuss sensitivity studies and trends; and Section 9 concludes the paper.

2. DRAM Architecture

In this section we briefly summarize the pertinent details of modern DRAM architectures. More complete descriptions can be found in related publications [6]–[8].

Modern DRAMs, such as SDR, DDR, DDR2, GDDR3, RDRAM, and XDR, are composed of a number of memory banks where each bank is a two-dimensional array. A location in the DRAM is accessed by an address that consists of bank, row, and column fields. For efficiency, row and column addresses are delivered through a single set of chip pins, and read and write data share a bi-directional data path from the pins to the sense amplifiers. Additionally, all the banks within a DRAM share the request and data paths. As a result of this minimalistic hardware, accessing data in a modern DRAM must adhere to the rules and timing constraints detailed below.

Since all the row and column fields of a DRAM address share the same data-path, multiple DRAM commands are required for accessing a particular location. First, an entire row within a specific DRAM bank is *activated* using a row-level command. Then, after the activation latency has passed, any location within that row can be accessed by issuing a column read or write command. *Internal bank conflicts* occur when two successive requests hit different rows of the same bank. In such an event, the current row must first be *precharged* before the new row can be activated. Hence, this second access requires two row-level commands (precharge and activate) followed by one column-level command (column read or write).

Table 1 shows key timing parameters of several DRAM generations. The table shows that as DRAMs evolve, latencies (in cycles) and access granularity are increasing. At the same time, DRAMs are becoming more sensitive to access

Symbol	Definition
AG	address generator
CB	channel-buffer
lB	number of words in a DRAM burst
lR	number of consecutive words requested by application (record-length)
MAS	memory access scheduler
MC	memory channel
mC	number of DRAM commands for accessing an inactive row
nAG	number of address generators
nB	number of internal DRAM banks
nCB	number of channel-buffer entries
nMC	number of memory channels
RB	memory reorder buffer
tCK	minimum time between two DRAM commands
$tdWR$	write to read turnaround cycle latency
tRC	activate-to-activate cycle latency within a single internal DRAM bank
tRR	activate-to-activate cycle latency across two internal DRAM banks
W	peak DRAM bandwidth (words per DRAM cycle)

Table 2: DPMS nomenclature

patterns as follows. Data pin bandwidth has been increasing rapidly while command bandwidth, specified by the number of cycles between two DRAM commands (tCK), has been increasing more slowly. This results in increasing access granularity, since each command must refer to a growing number of DRAM locations. Additionally we observe higher timing sensitivity to the actual access pattern. The time to read or write data on an idle bank ($tRCDW + tCWD$ for reads and $tRCDR + tCAC$ for writes) rises, requiring the memory system to be more deeply pipelined. Also, the interval between successive row activations to the same bank (tRC) has been increasing rapidly, making internal bank conflicts more costly. Even with no bank conflicts, the latency of consecutive row activations to different banks (tRR) has been growing, widening the gap between sequential-access and random-access performance. Finally, because internal data paths are shared between reads and writes, a write command followed by a read must be separated by a penalty of $tDWR+tWRBUB$ cycles ($tDRW+tRWBUB$ cycles for a read followed by a write). This bus turnaround time has also been growing over time, making DRAM performance more sensitive to interleaved reads and writes. Taken together, these trends show that DRAM performance is now very sensitive to the access pattern applied, leading to varied issues and tradeoffs in DPMS design, which we discuss in the next section.

3. Data Parallel Memory Systems

Data-parallel memory systems extract maximum throughput from modern DRAMs by exploiting parallelism and locality. Parallelism is utilized by pipelining memory requests to high-bandwidth DRAM components and also by interleaving accessing over multiple memory channels. DPMSs use *memory access scheduling* [9] to enhance locality by reordering memory accesses. This reordering improves performance by reusing open rows and by minimizing internal bank conflicts and read-write turnaround penalties.

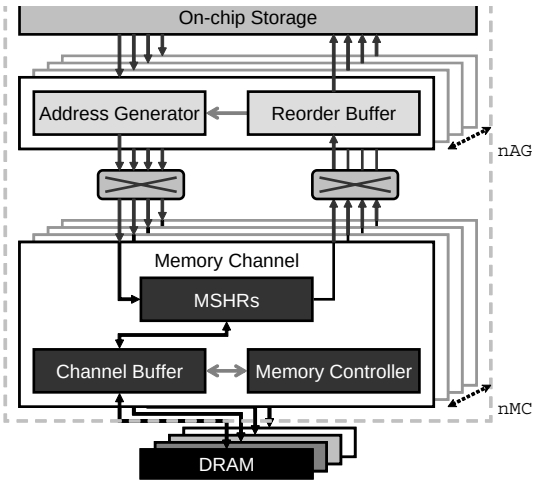


Figure 1: Canonical data-parallel memory system

Figure 1 shows a canonical DPMS that moves data between on-chip storage and one or more channels of off-chip DRAM. Table 2 summarizes the nomenclature used throughout the paper. Requests from the processor to perform memory operations are expanded into individual memory addresses by one or more address generators (AGs). To supply adequate parallelism, each AG is capable of generating multiple addresses per cycle, and several AGs may operate in parallel, with each generating accesses from different threads. Each access is routed to the appropriate memory channel (MC) via a crosspoint switch. The memory channel performs the requested accesses and returns read data to a reorder buffer (RB) associated with the originating AG. The optional RB collects and reorders replies from the MCs so they can be presented to the processor and on-chip storage in order. [1]

Within each MC, miss status holding registers (MSHRs) [10] act as a small non-blocking cache, keeping track of in-flight references and performing read and write coalescing of the requested accesses. The width of each MSHR entry is equal to the DRAM burst length. Accesses that cannot be satisfied by the MSHRs are forwarded to the *channel-buffer* to be scheduled. The memory controller schedules pending accesses in the channel-buffer, selecting one access during each DRAM command cycle, sending the appropriate command to the external DRAM, and updating the state of the pending access. Note that a single memory access may require as many as three DRAM commands to complete. The number of channel-buffer entries determines the size of the scheduling window used by the memory controller.

In the following paragraphs we examine the design options for each DPMS component, and identify critical issues that arise from DRAM technology trends.

[1] If a RB is not used, the on-chip storage must be capable of handling out of order replies from the memory system.

3.1. Address Generators

The design space of AGs consists of the number of AGs, the number of addresses each AG can generate each cycle (AG width), and the expressiveness allowed in the request streams.

Number of AGs: The number of AGs in a DPMS can be varied to trade off inter-thread parallelism for intra-thread parallelism. This tradeoff affects load balance across MCs and the locality of the resulting merged access stream. Several recently reported DPMSs [11], [12] use multiple AGs that can each generate a small number of references per cycle. This approach exploits parallelism mostly across memory reference threads, and improves load balance across MCs by combining references from multiple threads to even-out an unbalanced pattern from a single thread. Such access interleaving, however, reduces locality leading to decreased throughput as evidenced by the results in Sections 5 and Subsection 6.1. To avoid the loss of throughput due to such inter-thread interference, we suggest using a single wide AG, that generates many accesses per cycle, in place of multiple narrow AGs. This approach uses intra-thread parallelism in place of inter-thread parallelism to keep the memory pipeline full without sacrificing locality. However, it can suffer from MC load imbalance. In Subsection 3.4 we introduce a *channel-split* memory system organization that maintains both load balance and locality.

AG Width: The AG width (the number of accesses each AG can generate per cycle) determines the aggregate address bandwidth. When using multiple AGs, performance may be increased by providing more total AG bandwidth than the MCs are capable of handling. This allows the DPMS to maintain throughput even when the available thread parallelism is low. Providing excess AG bandwidth allows the memory system to adapt — exploiting intra-thread parallelism when it is available and falling back on inter-thread parallelism at other times.

AG Expressiveness: The simplest AGs handle only unit-stride access streams. To handle a wider range of applications, more sophisticated AGs handle both *strided* and *indexed* streams where each request is a sequence of consecutive words (i.e., a *record*). Strided access streams have a constant distance between records, whereas indexed (gather/scatter) streams make arbitrary record references.

3.2. Memory Access Scheduling

Memory access scheduling is used to reorder the DRAM commands associated with pending memory accesses to enhance locality and optimize throughput. We briefly explain MAS scheduling policies below and a more detailed description is available in [9].

The `inorder` policy makes no attempt to increase locality, issuing both row and column commands as they arrive. When the access pattern causes interference in DRAM, this policy results in very poor performance.

The `inorderla` policy issues column commands in order but looks ahead and issues row commands early, when they do not interfere with active rows or earlier pending column commands. This policy requires simple hardware and noticeably improves performance as shown in Subsection 6.3.

With the `firstready` policy, the oldest ready reference that does not violate DRAM timing constraints is issued to the DRAM. This reference may bypass older requests that are not ready to issue due to interference.

The `opcol`, `oprow`, `clcol`, and `clrow` policies all perform out-of-order scheduling to improve DRAM command locality. *Open* policies (`op*`) issue a row precharge command when there are no pending accesses to the row and there is at least one pending access to a different row in the same bank. *Closed* policies (`cl*`), on the other hand, precharge a row when the last available reference to that row is performed even if no other row within the bank is requested. Closed policies can take advantage of DRAM *auto-precharge* commands, which combine a precharge with a column read or write. When all else is equal, `*col` policies choose a column command, while `*row` give priority to row commands.

In this paper we show that while reordering references can lead to significantly improved DRAM throughput for indexed accesses, the specific scheduling policy chosen for reordering has only a second-order affect.

In addition to the scheduling policy, the channel-buffer size also affects performance. The deeper the buffer the greater the opportunity for the scheduler to discover and exploit locality. This is particularly important in the case of indexed reference patterns where successive references contain little locality.

3.3. Memory Channels

The design space of the memory channels includes the total number of MCs, the method of interleaving accesses across MCs, the number of MSHRs, and the channel-buffer depth.

Number of MCs: The number of MCs sets the peak memory bandwidth. More MCs provide more throughput at the expense of higher system cost. However, as mentioned above, care must be taken to ensure that the memory requests can be balanced among the MCs.

Address Interleaving Method: With multiple MCs, an interleaving policy is used to perform a mapping from a memory address to a MC. With direct interleaving, the MC is selected based on the low bits of the memory address. This simplistic distribution scheme can lead to transient load imbalance between channels when record lengths are large (Subsection 6.5). Pseudo-random interleaving, suggested in [13], alleviates this problem by basing the mapping on a hash of multiple address bits. In either case, the interleaving is done at the granularity of a DRAM burst.

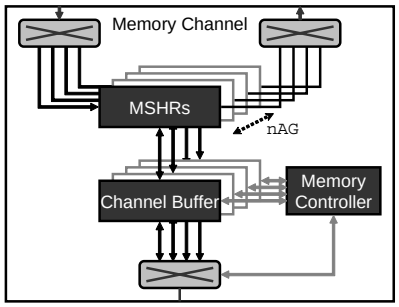


Figure 2: *channel-split* configuration

Internal MC Buffers: The channel-buffers and MSHRs must be deep enough to allow the scheduler to effectively deal with indexed accesses that display little inherent locality. We explore the performance sensitivity to this parameter in Subsection 6.4.

3.4. Channel-Split Configuration

We introduce a *channel-split* memory channel organization, shown in Figure 2, to exploit inter-thread parallelism to load balance across multiple MCs without reducing locality due to interference. In this organization, each AG has its own set of MSHRs and channel buffer entries in each MC. Dividing the resources by AG enables the scheduler to issue references from a single AG to maintain locality as long as the preferred AG has references to issue. If, due to load imbalance, the preferred AG has no references for the MC, the scheduler can select a reference from another AG to balance load. The channel-split organization improves performance, at the cost of increased channel-buffer entries and MSHRs. We discuss this further and show sensitivity results in Subsection 6.4.

This arrangement works particularly well to avoid read-write turnaround penalty. If one thread is performing read operations and a second thread is performing write operations, the scheduler will prefer scheduling reads from the first thread as long as they are available. The scheduler will only switch to performing writes when no further reads are available in the channel buffer. Note that with resources shared by all AGs, the second thread would eventually occupy all available buffers and the scheduler would be forced to issue a write, even if many reads were still available.

4. Experimental setup

To evaluate the DPMS design space and quantify the effects of DRAM technology trends, we measure the performance of several multimedia and scientific applications as well as targeted micro-benchmarks.

4.1. Test Applications

We use six applications from the multimedia and scientific domains summarized in Table 3. The scientific applications were slightly modified from their original implementation to enable them to run on the simulated configurations.

Name	Description
DEPTH	stereo depth extraction from two 320x240 images taken from different horizontal angles [11]
MPEG	encoding 3 frames of 360x288 video images according to the MPEG-2 standard [11]
RTSL	rendering the SPECviewperf 6.1.1 benchmark using the Stanford Real-Time Shading language [11]
MOLE	an n-body molecular dynamics solver [14]
FEM	finite element method code solving systems of first-order conservation laws on unstructured meshes [2]
QRD	converting a 192x96 complex matrix into an upper triangular and orthogonal matrices [11]

Table 3: Benchmark Programs - Applications

Our micro-benchmarks are designed to stress particular aspects of memory system performance. Benchmarks starting with a AxB format have strided accesses where the record length is A and the stride width is B. Benchmarks with a rA or crA prefix have indexed accesses where the index range is either 4M words (rA), or constrained to 64K words (crA), and A denotes the record length. Postfixes of the micro-benchmarks stand for the type of memory accesses and if consecutive streams can cause conflicts in DRAM when processed concurrently. Benchmarks named with rd contain only stream loads while rw benchmarks have stream loads and stores interleaved such that DRAM read-write turnaround penalty can be observed.[2] Benchmarks containing cf have stream loads and stores interleaved and these consecutive access threads hit different rows in the same DRAM bank to expose internal bank conflicts. Each micro-benchmark contains 8 access threads, and each thread accesses $4,096$ words, except for the 48x48rd benchmark, which accesses $3,840$ words per thread.

4.2. Simulated Machine Parameters

Simulations were performed using a cycle-accurate simulator for the Imagine stream processor [11], which has 8 SIMD clusters. Imagine uses a software managed *stream register file* (SRF) to stage data to and from external DRAM memory. The SRF is aligned with the clusters such that each cluster contains a processing element (PE) and a single SRF lane. The baseline configuration of the simulated Imagine and memory system is detailed in Table 4, and is used for all experiments unless noted otherwise.

In order to study DRAM trends and design space parameters in isolation of specific arithmetic execution characteristics, we treated all execution kernels as single-cycle operations. The memory access patterns and data dependencies in software managed data-parallel architectures do not depend on operation timing, which allowed us to obey all dependencies and maintain memory ordering in our applications. We chose to minimize arithmetic execution time in order to amplify the effect of memory system

[2]Note that stream, not record or word, loads and stores are interleaved. Each stream still contains either all loads or all stores.

Parameter	
Number of clusters	8
Operating frequency	1 GHz
Number of words in a DRAM burst (lB)	4
Peak DRAM bandwidth	4 GW/s
Number of words per DRAM row	1,024
Number of internal DRAM banks (nB)	8
Number of memory channels (nMC)	4
Number of addresses an AG can generate per cycle	4
Number of channel-buffer entries (nCB)	16
nCB for channel-split configurations	16 per AG
Number of words an MC can process per cycle	1
Memory access scheduling policy	`opcol`
DRAM timing parameters (Table 1)	XDR
Peak DRAM bandwidth(W)	2
Number of DRAM commands for accssing an inactive row (mC)	3

Table 4: Baseline machine parameters

performance trends. We demonstrate the amplification effect on memory system trends and verify the validity of our zero-cost computation evaluation methodology by also presenting results with realistic arithmetic timing.

5. Application Results

Figure 3 shows the throughput of the six applications on five simulated memory system configurations (bars, left axis). With four MCs operating at 1 GW/s each, peak bandwidth is 4 GW/s. The figure also shows the fraction of memory references that result in read-write switches and row commands (points, right axis). To explain the results of Figure 3 we refer to Table 5 which summarizes the memory access characteristics of the six applications.

Figure 3 shows that, except for the RTSL program, overall bandwidth is quite high — between 3.2 and 3.9 GW/s, 80% to 96% of the peak bandwidth. Table 5 shows that 35% of RTSL references are single-word indexed references over a large (200K word) address range. These essentially random references have little locality and hence result in low bandwidth — 1.2 GW/s. This lack of locality is also reflected in the high percentage of row commands for RTSL. The figure also shows that when the fraction of read-write switches and row commands increases, performance drops. Each read-write switch idles the DRAM during the turn-around interval, and row commands often result in bank conflicts as explained in Section 2.

The left-most bar in each graph shows the performance of a memory system with a long (32-word) DRAM burst length, 2 AGs, and in-order scheduling. The large lB gives poor performance on RTSL and MOLE because they make many indexed accesses to short records over a wide range. Hence, very little of the 32-word burst is used to transfer useful data. The next bar shows that reducing lB to 4 words, while improving performance on MOLE, reduces performance on most of the other applications. The smaller lB increases the number of row and column commands, since each command transfers less data, increasing interference and contention for command bandwidth. The center bar shows that this interference can be greatly reduced by using memory access scheduling (`opcol` policy) to enhance locality. This reordering significantly reduces the number of read-write switches and row commands, reducing idle time and bank conflicts respectively.

Even with `opcol` scheduling, DRAM bandwidth is limited by interference between the interleaved accesses from two AGs. The fourth bar shows how this interference can be reduced by using a single, wide AG. This reduction in read-write switches is most pronounced in QRD, which has a high fraction of writes (30%) and a large lR. This combination leads to competition over scarce buffering resources, limiting scheduler effectiveness and forcing read-write switches. Throughput can be improved by providing more buffering as discussed in Subsection 6.4.

MOLE has a small amount of MC load imbalance in each of its access threads, which results in a 4.2% reduction in performance when changing to a single AG. The right-most bar shows how the channel-split organization overcomes this load imbalance (bar 4) without sacrificing locality (bar 3) by dedicating MC resources to each AG. The channel-split configuration has the highest throughput for all of the applications except for RTSL for which it has 2.0% less throughput than the single-AG case. This anomaly is a result of the large number of relatively short stream-level memory operations of RTSL as discussed in Subsection 6.4.

Figure 4 presents memory system performance results with realistic arithmetic timing using eight ALUs per PE and shows the same basic trends as with zero-cost arithmetic, although much less pronounced. When using the baseline DRAM configuration with realistic arithmetic, as shown in Figure 4a, only MOLE and FEM require a significant portion of the 4 GW/s peak DRAM bandwidth. Therefore, there is virtually no observable performance difference for the other applications as the memory system configurations is varied. With a peak DRAM bandwidth of 0.4 GW/s (Figure 4b), which better matches application requirements, the trends demonstrated by DEPTH and RTSL also match the results of zero-cost arithmetic.

6. Sensitivity to DPMS Design Parameters

In this section we explore the sensitivity of throughput to DPMS design parameters, including the number of AGs, AG bandwidth, MC buffer depth, and scheduling policy. For each experiment, we take the baseline configuration described in Section 4 and vary one parameter while holding the others constant.

6.1. Number of AGs and Access Interference

Figure 5a shows throughput of the micro-benchmarks as the number of AGs is varied, for both conventional and

Application	Average strided record size (W)	Average strided stream length (W)	Average strided stride / record	Average indexed (W) record size	Average indexed (W) stream length	Average indexed (W) index range	Strided access	Read access
DEPTH	1.96	1802	1.95	1	1107	1180	46.6 %	63.0 %
MPEG	1	1515	1	1	1280	2309	90.1 %	70.2 %
RTSL	4	1170	4	1	264	216494	65.1 %	83.5 %
MOLE	1	480	1	9	3252	7190	9.9 %	99.5 %
FEM	12.4	1896	12.4	24	3853	203342	48.8 %	74.0 %
QRD	115	1053	350	N/A	N/A	N/A	100 %	69.0 %

Table 5: Application characteristics

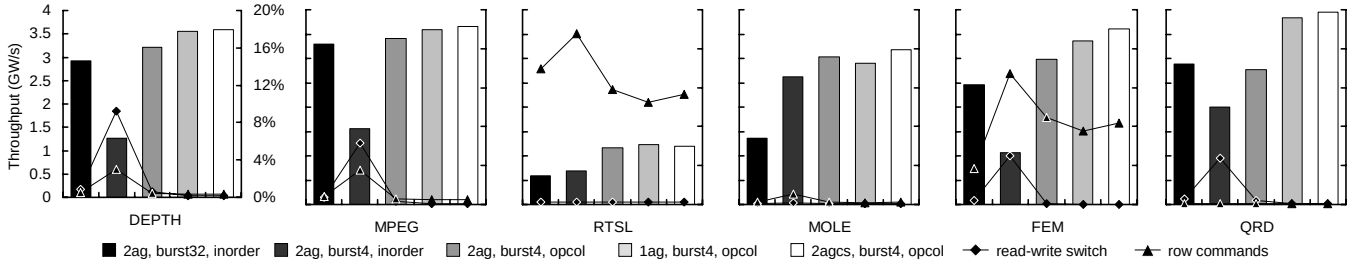


Figure 3: Memory system performance for representative configurations. Bars show throughput against left axis. Points show fraction of references of a particular type against the right axis.

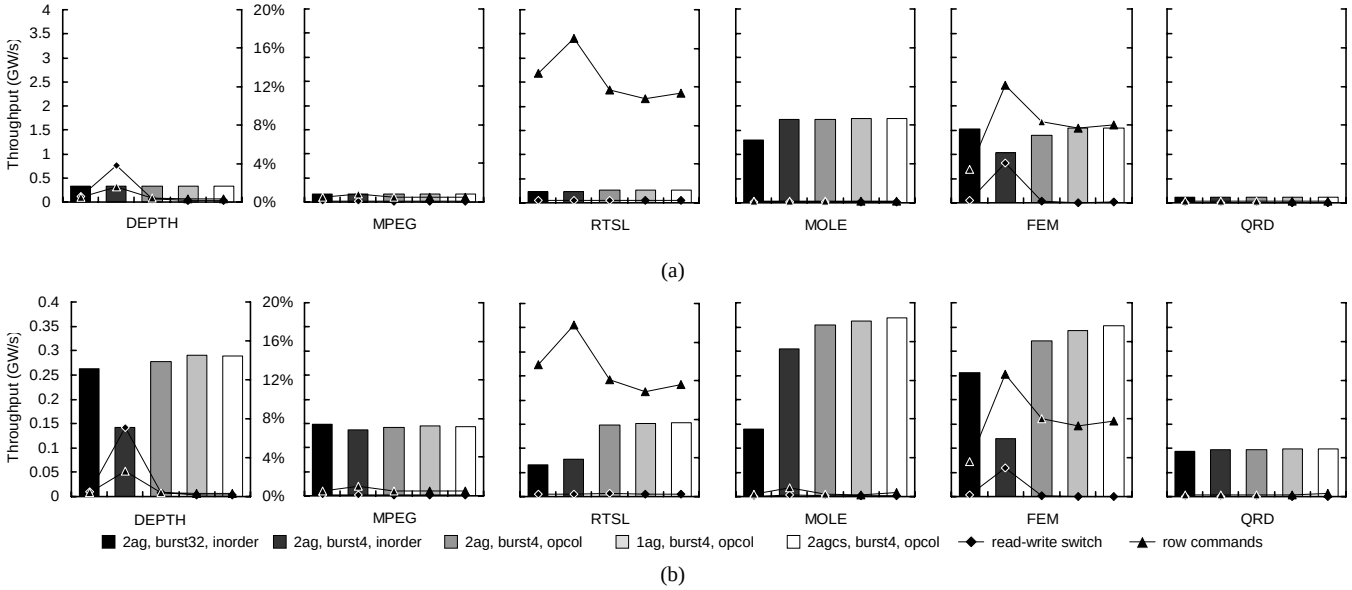


Figure 4: Memory system performance for representative configurations with realistic arithmetic timing and (a) 4 GW/s peak aggregate DRAM bandwidth, and (b) 0.4 GW/s peak DRAM bandwidth. Bars show throughput against left axis. Points show fraction of references of a particular type against the right axis.

channel-split organizations, while the number of channel-buffers is fixed to 64 per MC.

The channel-split configurations consistently give the highest throughput because they are able to achieve locality without sacrificing load balance. For benchmarks where no locality is present (cr1rd and r1rd), or when no interference exists (1x1rd), the number of AGs does not affect performance. Sequential access patterns with interference (1x1rw, 1x1rdcf, and 1x1rwcf) have poor performance with multiple AGs without channel splitting due to excessive read-write switches and row conflicts. Because of load imbalance, multiple AG configurations outperform the single-AG configuration on 48x48rd and r4rw.

Figure 5b shows application throughput as the number of AGs is varied. The results in the figure can be explained by matching the application characteristics (Table 5) with the micro-benchmarks. DEPTH and MPEG are dominated by sequential read and write accesses and therefore behave

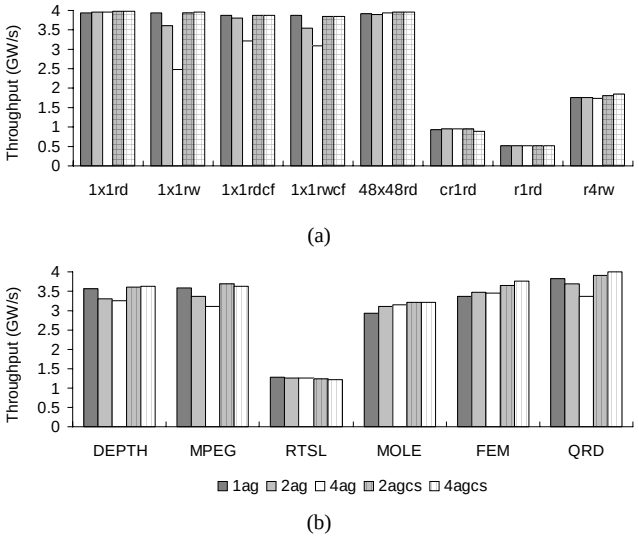


Figure 5: Throughput vs. AG configuration of micro-benchmarks (a) and applications (b)

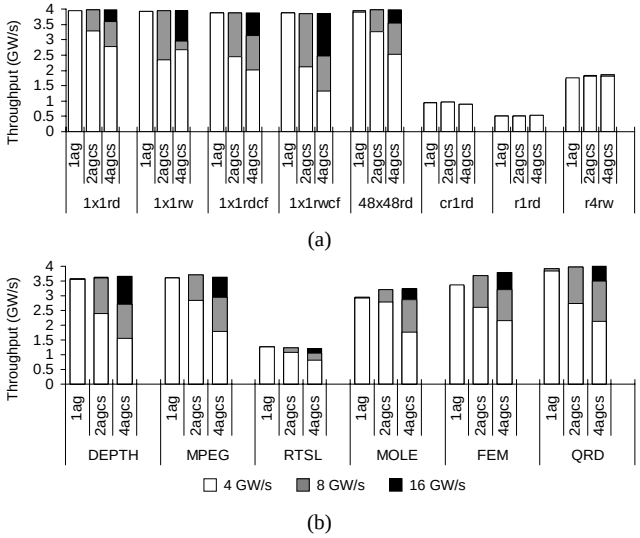


Figure 6: Throughput vs. total aggregate AG bandwidth of micro-benchmarks (a) and applications (b)

similarly to 1x1rw, 1x1rdcf, and 1x1rwcf. RTSL contains mostly reads with little locality and hence matches the behavior of r1rd. MOLE and FEM display similarities to r4rw as all three are dominated by indexed accesses to multi-word records, yet the channel imbalance of the real applications gives greater advantage to configurations with multiple AGs. QRD appears as a combination of 1x1rwcf and 48x48rd, since it has both large records and conflicting reads and writes characteristics.

The 4agcs configuration has 4% lower throughput than the 2agcs configuration on cr1rd, MPEG, and RTSL partly because of high channel-buffer pressure. We discuss this issue in more detail in Subsection 6.4.

6.2. Aggregate AG Bandwidth

Figure 6 shows throughput for micro-benchmarks and applications as the bandwidth of the AGs is varied and the number of channel-buffers is fixed to 64 per MC.

The figure shows that as the number of AGs is increased, higher aggregate AG bandwidth is needed to get maximum performance. A single AG achieves near-maximum performance when the AG bandwidth matches MC bandwidth (4 W/cycle). Increasing single AG bandwidth beyond this point provides at most a 2% performance advantage. Similarly, increasing the individual AG bandwidth beyond 4 W/cycle (e.g., aggregate bandwidth of 8 W/cycle on 2agcs) has no effect on effective DRAM bandwidth. For the multiple AG configurations, increasing aggregate bandwidth beyond the 4GW/s needed to saturate the MCs increases performance because it enables the scheduler to enhance locality by issuing all references from a single AG without causing a bottleneck. In benchmarks with little locality, such as cr1rd and r1rd, increasing aggregate AG bandwidth provides little return.

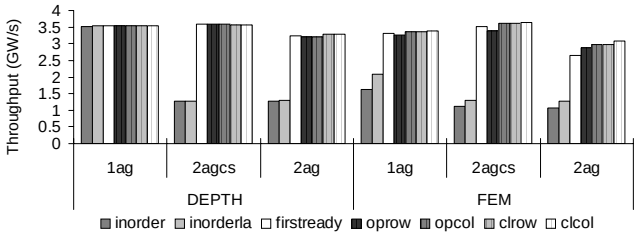


Figure 7: Throughput vs. memory access scheduling policy

6.3. Memory Access Scheduling

Figure 7 shows the sensitivity of throughput to scheduling policy for the DEPTH and FEM applications and three AG configurations. We present just two applications to avoid replication of data. DEPTH is representative of all strided applications and FEM is typical of all indexed applications. For strided applications, the single AG configuration performs well regardless of scheduling policy. With multiple AGs or indexed accesses, column reordering is needed to restore locality and achieve good performance. The throughput is largely insensitive to the exact column reordering policy used.

6.4. Memory Channel Buffers

Figure 8 shows that the sensitivity of throughput to the size of MC buffers rises as the number of AGs is increased. The single AG configuration reaches near-peak performance with only 16 channel-buffers, while even 64 buffers are not sufficient to achieve peak performance on the 4agcs configuration. Because the multiple AG configurations divide buffers across multiple threads, a single AG, even though it cannot load balance amongst MCs, gives higher performance when the total number of channel-buffers is limited. Because they can more flexibly allocate a scarce resource, non-split

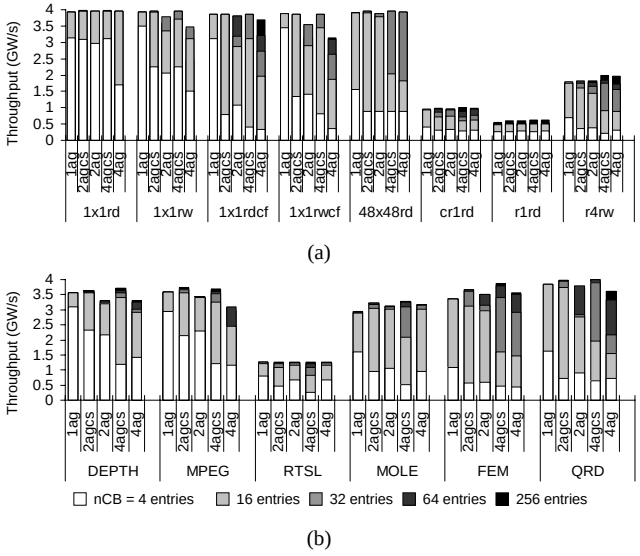


Figure 8: Throughput vs. total MC buffer depth of micro-benchmarks (a) and applications (b)

configurations outperform the channel-split approach when we limit channel-buffer depth to under 16 entries per MC.

When enough buffering is provided, the locality-preserving configurations perform well, and providing load balancing through multiple AGs can improve throughput. The largest performance improvements are seen for DEPTH, FEM, and QRD where 4agcs outperforms 1ag by 4.2%, 15%, and 6.8% respectively. 1ag, however, has a 2% higher throughput than the channel-split configurations on RTSL. This minor difference is the result of the large number of stream-level memory operations of RTSL, combined with the increased latency of each operation caused by the deeper buffering required to support the channel-split mechanism.

Additionally, we observe diminishing returns from increasing channel-buffer depth beyond 16 entries per AG. In 2agcs, for example, increasing nCB from 32 to 64 entries results in at most 3.5% and 9.4% gains in throughput for the applications and micro-benchmarks respectively (RTSL and cr1rd).

Due to the potential higher cost, in terms of area and/or cycle-time pressure, of a large number of channel-buffer entries and high AG width, we advocate the use of the 1ag configuration with 16 channel-buffer entries. If higher performance is desired, 2agcs with a total buffer depth of 32 can reach throughput that is within 6.3% of the maximum throughput of any configuration across all six applications.

6.5. Address Interleaving Method

The method used to map addresses to MCs in an interleaved DPMS can have a significant effect on the amount of channel buffering required to achieve optimal performance. When a direct method is used, transient load imbalance between MCs can occur, requiring deeper buffers to prevent stalls. As shown in Figure 9, even with $nCB = 16$, the effect is quite small (less than 3.3%) for DEPTH, MPEG, RTSL, and MOLE. FEM and QRD have a greater disparity unless deep channel buffers are used (19.4% with $nCB = 16$ and 1.7% with $nCB = 64$ for QRD). The reason for this sharp difference is the clustered access pattern resulting from the very long records of QRD. The simulated stream processor contains 8 SIMD PEs and each PE processes an entire record. When the records are large, the generated access pattern is sets of 8 words separated by multiples of the stride. When the low-order address bits directly determine the MC, each group of 8 accesses is mapped to the same MC, leading to a transient imbalance. If the channel buffers are not deep enough to tolerate such transients, the AGs must stall and wait for buffers to be freed as accesses complete. A pseudo-interleaved mapping decorrelates the mapping from the number of PEs and alleviates the temporary buffer pressure.

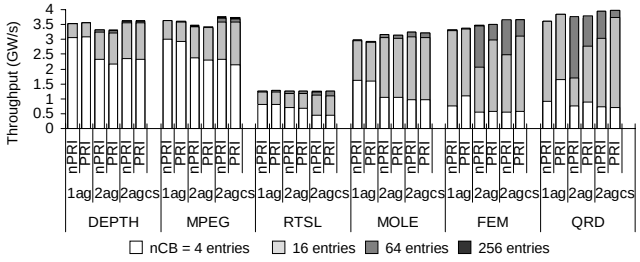


Figure 9: Throughput vs. address interleaving method — (n)PRI = (no) Pseudo Random Interleaving

7. Sensitivity to DRAM Technology Trends

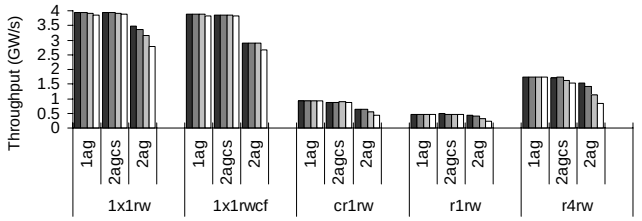
In this section we explore the effects of extending the trends of increasing DRAM latencies and burst lengths. We also examine the implications of limited DRAM command issue bandwidth.

7.1. Read-Write Turnaround Latency

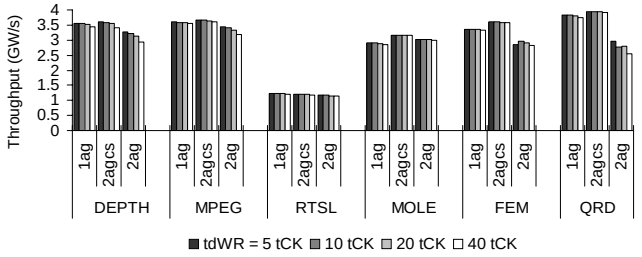
As DRAM latencies increase performance becomes even more sensitive to access patterns and locality is even more critical to maintain throughput. Figure 10 shows that the locality-conserving 1ag and 2agcs configurations lose only 5.2% throughput as $tdWR$ is increased from 5 to 40 tCK (DEPTH). When locality is sacrificed in the 2ag configuration, performance drops by up to 46% (r4rw). With some cases of MOLE, FEM and QRD, performance slightly increases as latency is increased in the 2ag configuration. This anomalous behavior is a result of scheduling artifacts when switching between read and write threads. Due to the greedy nature of the scheduling algorithm, increasing $tdWR$ occasionally gives a better schedule.

7.2. DRAM Burst Length and Indexed Accesses

Figure 11a presents the results of increasing DRAM burst length for several micro-benchmarks. Changing lB has no effect on 1x1rd. The 1x1rw benchmark shows that 1ag and 2agcs, which maintain locality even when both reads and writes exist, are also insensitive to the value of lB. The

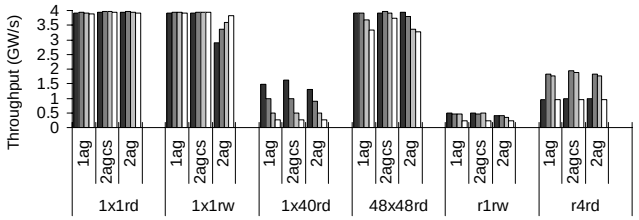


(a)

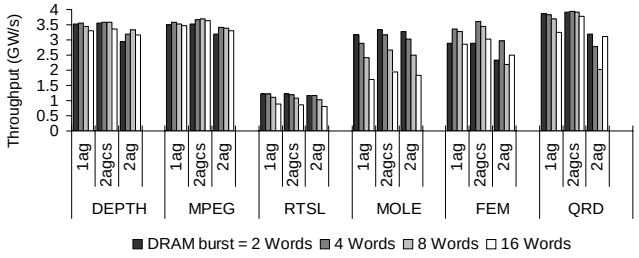


(b)

Figure 10: Throughput vs. DRAM $tdWR$ of micro-benchmarks (a) and applications (b)



(a)



(b)

Figure 11: Throughput vs. DRAM burst length of micro-benchmarks (a) and applications (b)

throughput of 2ag increases with burst length (from 2.9 to 3.8 GW/s) because the number of read-write switches is reduced (see also Section 5). The performance of 1x40rd drops rapidly with rising lB because the large stride of this benchmark results in only a single usable word from each burst. Thus, the throughput drops by a factor of 2 each time we increase the burst length by a factor of 2. The throughput of 48x48rd also drops as lB grows. This behavior is a result of timing sensitivities in channel buffer and MSHR management. A detailed examination of the behavior shows that MSHRs are being deallocated while there are still matching accesses in the channel buffers, leading to excess consumption of DRAM bandwidth and a total drop of 15% in throughput with the 1ag and 2ag configurations. Random access patterns work best with a burst of 4 or 8 words (r1rw and r4rd).

Trends are less clear in the case of the multimedia and scientific applications, which have more complicated access patterns than the micro-benchmarks (Figure 11b). Different access patterns work better with different values of lB, yet we observe a sweet spot with a DRAM burst of 4 words. Both the 1ag and the 2agcs configuration display near optimal performance with $lB = 4$, with the exception of MOLE where a burst of 2 words achieves 4.7% higher throughput with the 2agcs configuration.

To better understand the effect of changing the DRAM burst length, we develop an analytical model for the expected throughput based on DRAM technology parameters. Please refer to Table 2 in Section 3 for a description of the symbols used below.

In order to keep the formula for effective random indexed bandwidth concise we make the following assumptions:

- All accesses are indexed, with the indices uniformly distributed over the entire DRAM address space.
- All accesses are to a fixed record length (lR).
- A record can be accessed by issuing at most one column command to each memory channel ($lR \leq lB \cdot nMC$).

If the index space is sufficiently large, these assumptions result in every record access requiring a new row command to an inactive bank. This leads to the following formula for the effective DRAM bandwidth[3]:

$$BW_{rand} \simeq BW_{peak} \frac{lR}{\lceil \frac{lR}{lB} \rceil lB_{effective}} \tag{1}$$

The effective burst length ($lB_{effective}$) is given by:

$$lB_{effective} = W \cdot \max\left(\frac{lB}{W}, mC, tRR, \frac{tRC}{nB}\right) \tag{2}$$

The four arguments of the $\max$ function in Equation (2) correspond to four factors that may limit the rate of record accesses. The first term is the actual burst length and remaining arguments all relate to command bandwidth. It takes mC commands to access each record, so at least ($W \cdot mC$) words must be transferred on each access, or command bandwidth is the bottleneck. The third term is due to the maximum bandwidth of accesses to different banks. Finally, when tRC is large compared to nB at most one access can be made each $\frac{tRC}{nB}$ cycles.

We can now write the full equation for random throughput:

$$BW_{rand} \simeq BW_{peak} \frac{lR}{\lceil \frac{lR}{lB} \rceil \cdot W \cdot \max\left(\frac{lB}{W}, mC, tRR, \frac{tRC}{nB}\right)} \tag{3}$$

Figure 12 compares our analytical model to simulation results for the r1rd and r4rd micro-benchmarks. We vary nB and lB, holding all other parameters to the baseline described in Section 4. In all cases the model matches well

[3] $\lceil \frac{lR}{lB} \rceil$ is the number of bursts required to access a record

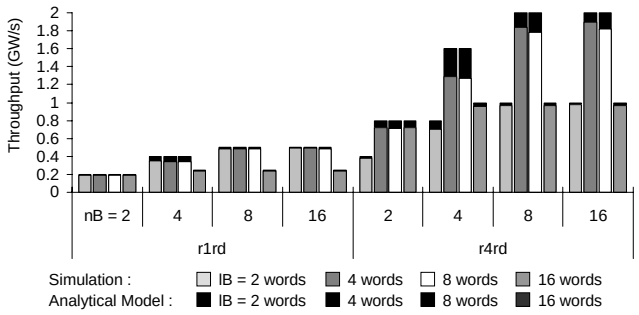


Figure 12: Analytical model and measured throughput of random indexed access patterns

with simulated performance and is within 8.2% of the measured effective random bandwidth on average. The relatively large 17% error in the modeled throughput for r4rd with 4 internal DRAM banks is a result of not accurately modeling contention for DRAM command issue. With complex DRAM timing constraints and memory access scheduling, the restriction of issuing only one DRAM command per cycle to each MC can become a temporary bottleneck.

We can see the detrimental effect of limited command issue bandwidth when the number of internal banks is small. With $tRC = 20$ and only 2 or 4 banks, a new access can be made at most once every 10 or 5 cycles, which limits performance. Additionally, when $lB = 2$, two commands are required for each access in the case of r4rd, and again, performance is sacrificed. The problem of long bursts is evident when $lB = 16$ and much of the DRAM bandwidth is squandered transferring unused data.

8. Related Work

Traditional vector processors such as the CRAY-1 [1] and the CRAY X-MP [15] have memory systems with parallel, interleaved memories allowing concurrent memory accesses between multiple vector streams [16]. Multiple accesses, especially from multiple streams, often result in memory bank conflicts. Prior work [13], [17], [18] analyzed these bank conflicts and suggested techniques to alleviate performance degradation in the context of strided accesses to SRAM. One of the random interleaving schemes of [13] is implemented in the memory systems evaluated in this paper. Kozyrakis [19] studied memory system design space in embedded vector processors concentrating on memory channel and internal memory bank configurations for embedded DRAM technology. Our work, in contrast, focuses on multi-word strided and indexed accesses with both inter- and intra-thread access reordering to modern DRAMs.

The Stream Memory Controller [20], the Command Vector Memory System [21], and more advanced Memory access scheduling [9] studied hardware/software techniques for streaming accesses by exploiting the three-dimensional nature of DRAMs. However, these works did not cover the interaction of concurrent streaming access threads and the corresponding design choices in depth. Specifically, [9] was limited in scope to evaluating a specific choice of DRAM parameters (first generation SDRAM specification) and to a single technique of reordering memory operations to increase throughput — memory access scheduling. This technique is just one of eight different design space parameters explored in this paper. The more thorough exploration of DRAM property trends, based on the analysis of multiple DRAM families and future projection, lead us to significantly different conclusions than prior work. For example, contrary to [9], we conclude that the exact column/row reordering policy has only a second order effect on memory throughput. Additionally, the recent increase in read-write turnaround latencies affects performance as strongly as column/row locality, leading us to recommend a different design point than the one suggested in prior work.

9. Conclusion

In this paper we explored the design space of data-parallel memory systems in light of the DRAM technology trends of rapidly increasing DRAM bandwidth and a very slow improvement in DRAM latency. We have shown that these trends lead to growing DRAM access granularity and high sensitivity of throughput to application access patterns. Specifically, we identified the importance of maintaining locality in the reference pattern applied to the DRAM in achieving high performance. Locality is critical to reduce the total number of DRAM commands and the amount of time spent idling due to internal DRAM bank conflicts and read-write turnaround penalties.

We presented a detailed taxonomy of the DPMS design space, and examined two hardware configurations that are able to exploit locality. The first is a single, wide address generator architecture that operates on one access thread or stream at a time. The single-AG configuration takes advantage of locality, however, the reliance on only a single thread can lead to load imbalance between memory channels, and hence, to reduced performance. Our experimental results indicate that the main advantages of the single-AG approach are in allowing the memory access scheduler to process all operations of a single type (reads or writes) before having to sustain a read-write turnaround penalty, and in limiting the number of internal DRAM bank conflicts.

Based on these observations, we designed the novel channel-split hardware mechanism that is able to achieve locality and balance load across multiple channels at the same time. The channel-split configuration dedicates buffering resources within each memory channel to a specific AG. In this way the scheduler is free to pursue locality within a single thread without competition for buffers. When one thread stalls, however, references from other threads can proceed to balance the load. The hardware costs of achieving both locality and load balance, are increasing the buffering resources to account for the hard partition, and providing excess address-generation bandwidth to maintain performance at times of low thread parallelism.

We performed comprehensive sensitivity analysis and showed that with a single, wide AG and only 16 channel-buffer entries we achieve over 80% of peak performance on most applications. Dedicating additional hardware resources increases performance by up to 7.4% with a 4-AG channel-split configuration that generates an aggregate of 16 addresses per cycle and relies on 256 channel-buffer entries in each MC. A less costly configuration with 2 AGs and 32 channel-buffer entries per MC improves performance by up to 3.6% over using a single AG.

Finally, we explored the impact of increasing DRAM latency and granularity. We show that conventional multiple-AG configurations suffer when read-write turnaround delays are increased. Locality-conserving single-AG and channel-split configurations, however, maintain throughput even when the latency is increased eightfold. We also quantified the detrimental effects of coarser-grained DRAM commands and reduced effective DRAM command issue rate on random-access reference patterns and developed an accurate analytical model for the expected throughput.

References

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

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

[3] J. Ahn and W. J. Dally, "Data Parallel Address Architecture," *Computer Architecture Letters*, vol. 4, 2005.

[4] Samsung Electronics, "512Mbit SDRAM Specification," http://www.samsung.com/Products/Semiconductor/DRAM/.

[5] ——, "512Mbit DDR2 SDRAM," http://www.samsung.com/Products/Semiconductor/DRAM/.

[6] ——, "512Mbit GDDR3 SDRAM," http://www.samsung.com/Products/Semiconductor/GraphicsMemory/.

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

[8] B. T. Davis, "Modern DRAM Architectures," Ph.D. dissertation, The University of Michigan, Ann Arbor, 2001.

[9] 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*, Vancouver, B.C., Canada, Jun 2000, pp. 128–138.

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

[11] J. 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*, Munich, Germany, Jun 2004, pp. 14–25.

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

[13] B. R. Rau, "Pseudo-Randomly Interleaved Memory," in *Proceedings of the 18th Annual International Symposium on Computer Architecture*, Toronto, Ontario, Canada, May 1991, pp. 74–83.

[14] M. Erez, J. Ahn, A. Garg, W. J. Dally, and E. Darve, "Analysis and Performance Results of a Molecular Modeling Application on Merrimac," in *SC'04*, Pittsburgh, Pennsylvaniva, Nov 2004.

[15] M. C. August, G. M. Brost, C. C. Hsiung, and A. J. Schiffleger, "Cray X-MP: The Birth of a Supercomputer," *IEEE Computer*, vol. 22, no. 1, pp. 45–54, Jan 1989.

[16] T. Cheung and J. E. Smith, "A simulation study of the CRAY X-MP memory system," *IEEE Transactions on Computers*, vol. 35, no. 7, pp. 613–622, 1986.

[17] G. S. Sohi, "High-Bandwidth Interleaved Memories for Vector Processors - A Simulation Study," *IEEE Transactions on Computers*, vol. 42, no. 1, pp. 34–44, 1993.

[18] W. Oed and O. Lange, "On the Effective Bandwidth of Interleaved Memories in Vector Processor Systems," *IEEE Transactions on Computers*, vol. 34, no. 10, pp. 949–957, 1985.

[19] C. Kozyrakis, "Scalable Vector Media-processors for Embedded Systems," Ph.D. dissertation, University of California, Berkeley, 2002.

[20] S. I. Hong, S. A. McKee, M. H. Salinas, R. H. Klenke, J. H. Aylor, and W. A. Wulf, "Access Order and Effective Bandwidth for Streams on a Direct Rambus Memory," in *Proceedings of the 5th International Symposium on High Performance Computer Architecture*, Orlando, Florida, 1999, pp. 80–89.

[21] J. Corbal, R. Espasa, and M. Valero, "Command Vector Memory Systems: High Performance at Low Cost," in *Proceedings of the 7th International Conference on Parallel Architectures and Compilation Techniques*, Paris, France, 1998, pp. 68–77.