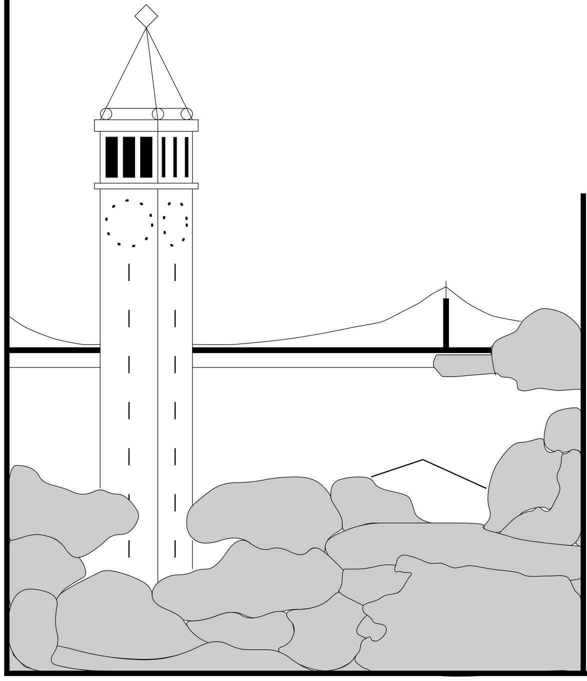
Vector IRAM Memory Performance for Image Access Patterns

Richard M. Fromm



Report No. UCB/CSD-99-1067

October 1999

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Vector IRAM Memory Performance for Image Access Patterns

by Richard M. Fromm

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor David A. Patterson
Research Advisor

(Date)

* * * * * * *

Professor Katherine Yelick
Second Reader

(Date)

Abstract

The performance of the memory system of VIRAM is studied for various types of image accesses representative of multimedia applications. The performance of VIRAM-1 and other variations on the VIRAM architecture are characterized. The mean bandwidth for loading images of various sizes for the default VIRAM configuration is 6.4 GB/s for a horizontal image access pattern, 0.38 GB/s for a vertical image access pattern, 1.4 GB/s for an 8 × 8 blocked image access pattern, and 0.20 GB/s for a random image access pattern. For stores, the mean bandwidth is 6.4 GB/s, 0.19 GB/s, 1.1 GB/s, and 0.10 GB/s for horizontal, vertical, 8 × 8 blocked, and random image access patterns, respectively. These compare to peak bandwidths of 6.4 GB/s, 0.8 GB/s, 6.4 GB/s, and 0.8 GB/s for the horizontal, vertical, 8 × 8 blocked, and random image access patterns, respectively.

Averages can be deceiving, however, as there is sometimes a wide variance amongst the results. This phenomena is especially true for strided accesses, found in the vertical image access pattern, whose performance is highly dependent on the stride. Hardware-based data layout alternatives are examined for their effect on strided memory performance. An alternative layout modestly improves the mean performance of the vertical access pattern, but it increases the variance and decreases the performance of some particular cases. A simple address hashing scheme decreases the variance and increases the performance of some particular cases, but it decreases the mean performance of the vertical access pattern.

The bottlenecks to performance within the memory system are sometimes bank conflicts, sometimes sub-bank conflicts, and sometimes a mixture of the two. When sub-bank conflicts are a significant factor, the performance significantly increases if each bank within the DRAM is divided into sub-banks, and load bandwidth is higher than store bandwidth due to the additional sub-bank busy time for stores. Other factors limiting the performance of the VIRAM memory system include short vectors, insufficient issue bandwidth, and the effects of a simplified pipeline control. Loop unrolling is necessary for maximizing performance when there is insufficient issue bandwidth to keep one or both memory units busy, in the horizontal and blocked image access patterns. Data alignment is only significant on unit stride accesses when there is sufficient issue bandwidth to keep the vector memory unit(s) busy.

The memory system is a limiting factor in the ability of the vector unit to effective scale both the number of lanes and the number of address generators. Scaling improves as the number of sub-banks increases for cases in which sub-bank conflicts are a significant factor.

Even though there are limitations to scaling and all but the unit stride accesses of the horizontal image access pattern achieve less than the peak performance, the absolute performance of VIRAM-1 is impressive compared to conventional, cache-based machines. For comparison, the measured unit stride performance of a memory to memory copy on a PC running at twice the clock frequency of VIRAM-1 is only 304.0 MB/s [20], a small fraction of the sustainable unit stride bandwidth of VIRAM-1.

Contents

List of Figures

Section 1

Introduction

Hardware trends, driven by the infamous and ever-growing processor-memory performance gap [9], are leading to the convergence of processors and memory. Such a convergence gives the potential for a reduction in memory latency as well as a huge increase in bandwidth, and a challenge is finding an architecture that can best take advantage of this. Vector architectures are a good match for the hardware characteristics of Intelligent RAM (IRAM), as they are able to efficiently take advantage of a large memory bandwidth.

Software trends are pointing to an era in which traditional desktop PC applications will not be dominant. Multimedia and embedded applications, and sensitivity to low power requirements, will become increasingly important [4][6]. Such data-parallel applications are often highly vectorizable; vectors are not merely limited to scientific computing. Vector architectures also have advantages for low power consumption when compared to conventional superscalar machines [2]. The regularity of vector implementations will scale well to the future, as wiring delays become more significant with respect to logic gate delays and control complexity and verification issues become even more important than they already are.

The IRAM project at UC Berkeley is designing and implementing VIRAM-1, a single chip vector microprocessor integrated with a scalar core and DRAM main memory. The Vector IRAM (VIRAM) instruction set architecture is an extension to the MIPS-IV ISA [13]. Along with an assembler and instruction-level simulator, a near cycle-accurate performance simulator has been developed, `vsim-p`, to assist in application development, provide feedback to the microarchitecture development, and investigate ideas beyond the scope of the VIRAM-1 implementation.

Previous studies of VIRAM [15] have examined the performance of computationally intensive kernels. This study uses `vsim-p` to study the performance of the memory system for various types of accesses representative of multimedia applications. Besides characterizing the performance of VIRAM-1 for a set of memory microbenchmarks, it explores the following issues:

- Using `vsim-p` to optimize performance through coding changes, such as data alignment and loop unrolling.

- Investigating aspects of the VIRAM-1 microarchitecture that are still in question, such as alternative data layouts and hashing address interleaving schemes.

- Identifying where the memory bottlenecks are for various scenarios, including determining the importance of multiple sub-banks per DRAM bank.

- Studying how well the architecture scales, both as the number of lanes is scaled down and up and the number of address generation resources is increased.

This study assumes that the reader already has significant familiarity with the IRAM project and VIRAM-1. [17] and [16] give a good overview of the IRAM project. Details about the VIRAM instruction set can be found in the ISA manual [18], and details about the VIRAM-1 implementation can be found in the microarchitecture manual [14]. Much more information about `vsim-p`, including a software perspective of the VIRAM-1 implementation and a more generic description of the VIRAM architecture, including ways in which the software tools allow it to parameterized, is given in the documentation for the performance simulator [8].

The rest of this paper is organized as follows. Section 2 reviews the microarchitecture of the VIRAM memory system. Section 3 describes the micro-benchmarks used in this study. Section 4 summarizes the metrics used for evaluating the performance. Section 5 describes the methods used to obtain results. Section 6 presents the detailed results from running the various micro-benchmarks for a variety of hardware and software configurations. Section 7 finishes with conclusions and thoughts for future work.

Section 2

VIRAM Memory Microarchitecture

Before going into more details about the studies performed, it is useful to briefly review the microarchitecture of the VIRAM memory system.

2.1 Organization

The memory system in VIRAM is divided into a multi-level hierarchy. Figure 2.1 shows the default VIRAM memory configuration. The memory is divided into 2 wings; each wing consists of 8 banks; and each bank is divided into 8192 rows of 2048 bits each. This gives a total memory size of 32 MB.[1] The 2048 bits within each row make up the DRAM page, the unit of granularity read from the DRAM array into the sense amps. The row is further divided into 8 columns of 256 bits each. The smallest unit with which the DRAM can be addressed, from the perspective of the memory controller, is 64 bits. Each column therefore spans 4 64-bit accesses. In this study, the data being loaded and stored is 8-bit pixel data. The minimum unit of access to the DRAM therefore spans 8 pixels.

It is possible for each DRAM bank to be organized into independent sub-banks. Figure 2.2 shows the details within a single wing for a configuration with 4 sub-banks. In the default VIRAM configuration, however, each bank consists of only a single sub-bank, as shown in Figure 2.1.

The layout of a program's data in memory is governed by the placement of the fields in the address decode that correspond to the various levels of the memory. The hardware data layout is expressed by a 5 character string, where the letters W, B, S, R, and C (for wing, bank, sub-bank, row, and column respectively) each appear exactly once. Bits after the W, B, S, R, and C fields in the address decode determine the offset for the bytes within a column. The order in which the bits are interpreted, from MSB to LSB, corresponds to the order of the characters in the string, from left to right. The default VIRAM layout is RSBCW.

Given this layout, the numbered labels on Figures 2.1 and 2.2 show the order in which data is accessed for each of the configurations as the physical address is increased. Within a 256-bit column (the offset field), the 4 64-bit accesses are always ordered one after another, as are the 8 8-bit bytes within an access. Successive column accesses alternate between the 2 wings (W) until all of the columns (C) within the DRAM pages in

[1]The memory configuration of the VIRAM-1 implementation has recently been reduced from 8 to 4 banks per wing, for a total of 16 MB, due to a lower than expected DRAM density from our industrial partner. This study assumes the original 32 MB configuration. While this change may slightly shift the boundary positions demarcating regions where certain types of conflicts occur, the overall conclusions should remain the same.

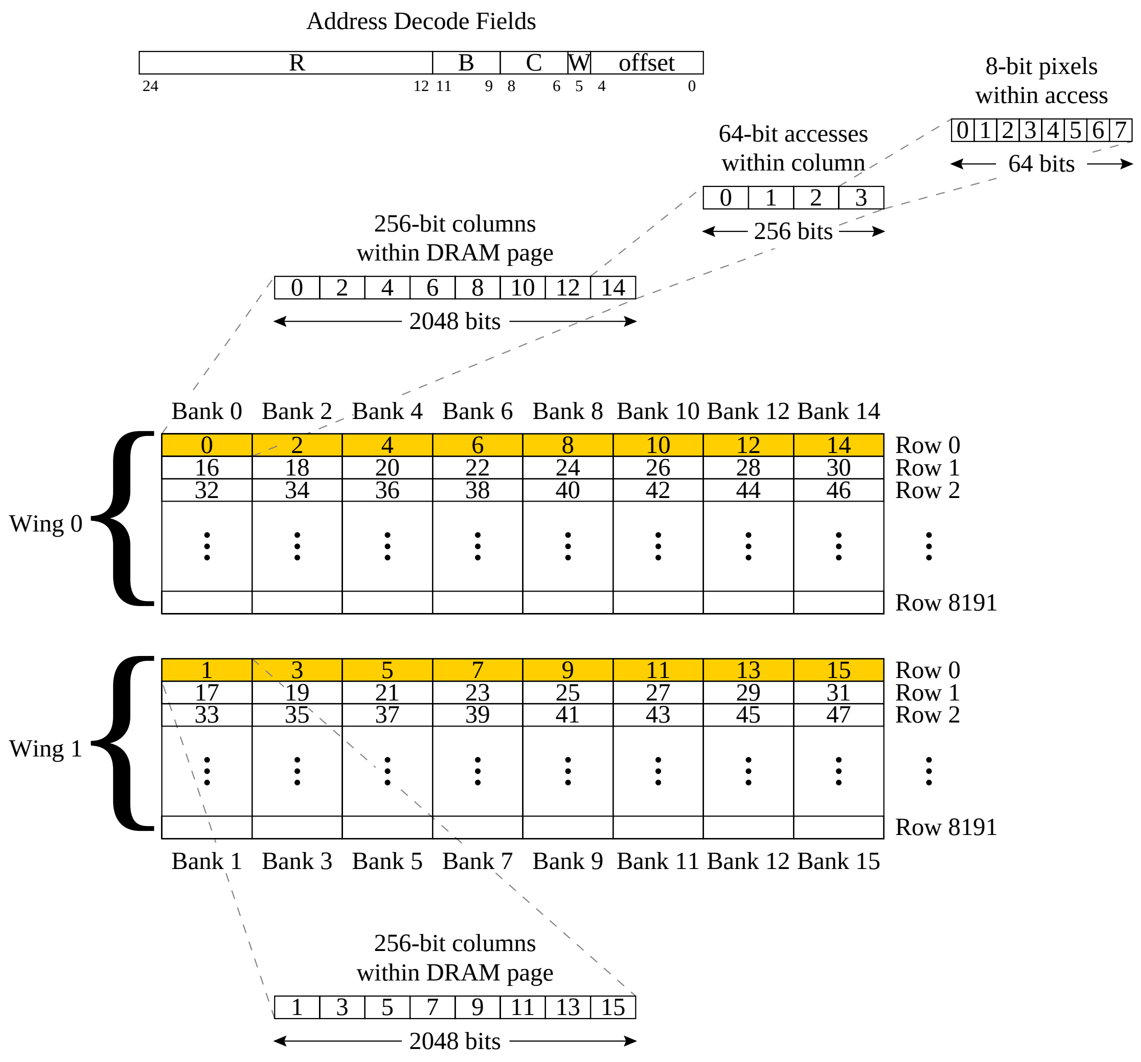

Address Decode Fields
R
B
C
W
offset
24
12 11
9 8
6 5 4
0
8-bit pixels within access
0 1 2 3 4 5 6 7
64 bits
64-bit accesses within column
0 1 2 3
256 bits
256-bit columns within DRAM page
0 2 4 6 8 10 12 14
2048 bits
Bank 0 Bank 2 Bank 4 Bank 6 Bank 8 Bank 10 Bank 12 Bank 14
0 2 4 6 8 10 12 14 Row 0
16 18 20 22 24 26 28 30 Row 1
32 34 36 38 40 42 44 46 Row 2
Row 8191
Wing 0
1 3 5 7 9 11 13 15 Row 0
17 19 21 23 25 27 29 31 Row 1
33 35 37 39 41 43 45 47 Row 2
Row 8191
Wing 1
Bank 1 Bank 3 Bank 5 Bank 7 Bank 9 Bank 11 Bank 12 Bank 15
256-bit columns within DRAM page
1 3 5 7 9 11 13 15
2048 bits

Figure 2.1: Default VIRAM Memory Configuration. The 2 wings, 8 banks per wing, and 8192 rows of 2048 bits each give a total memory size of 32 MB. Numbers within the blocks refer to the order of accesses for the default layout of RSBCW. There is no sub-bank (S) address decode field because there is only 1 sub-bank per bank. The highlighted row is used for illustrative purposes at several points within the text.

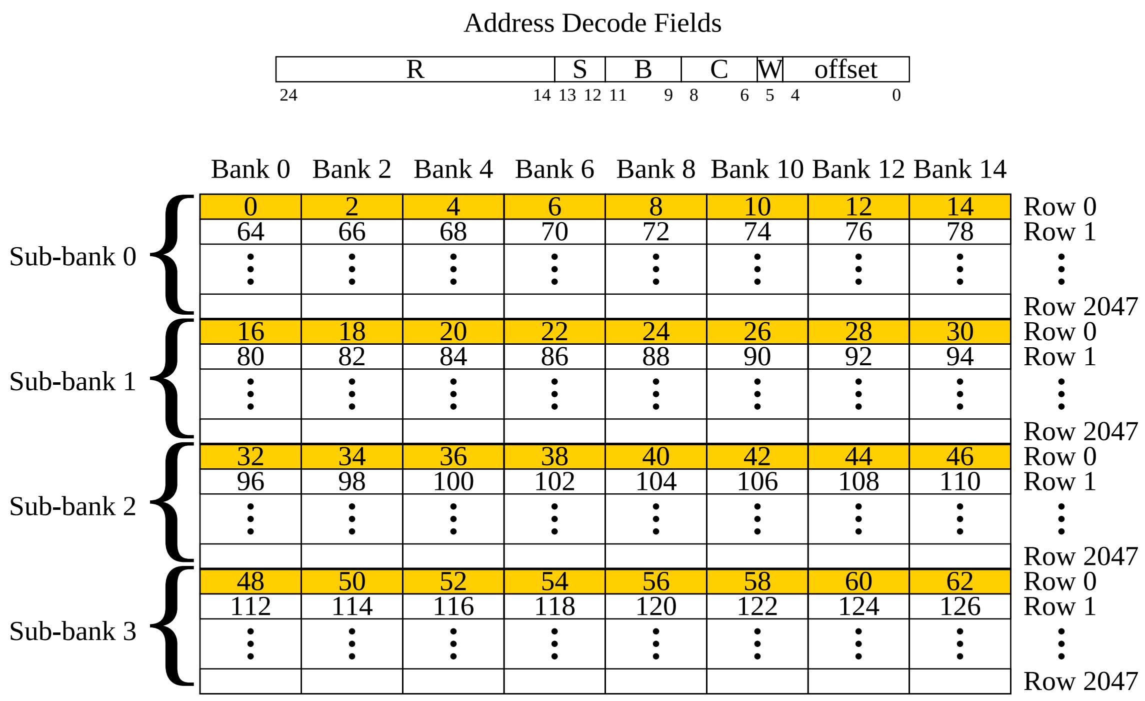

Address Decode Fields
R
S
B
C
W
offset
24
14 13 12 11
9 8
6 5 4
0
Bank 0 Bank 2 Bank 4 Bank 6 Bank 8 Bank 10 Bank 12 Bank 14
Sub-bank 0
0 2 4 6 8 10 12 14 Row 0
64 66 68 70 72 74 76 78 Row 1
Row 2047
Sub-bank 1
16 18 20 22 24 26 28 30 Row 0
80 82 84 86 88 90 92 94 Row 1
Row 2047
Sub-bank 2
32 34 36 38 40 42 44 46 Row 0
96 98 100 102 104 106 108 110 Row 1
Row 2047
Sub-bank 3
48 50 52 54 56 58 60 62 Row 0
112 114 116 118 120 122 124 126 Row 1
Row 2047

Figure 2.2: Effect of Sub-banks. A single wing from Figure 2.1 is shown divided into 4 sub-banks. (This is *not* the default configuration.) Numbers within the blocks refer to the order of accesses for the default layout of RSBCW. Odd numbered blocks map to the opposing wing (not shown). The highlighted row is used for illustrative purposes at several points within the text.

the open rows in both wings have been consumed. Then the addressing advances to the next bank (B). Once all of the banks have been covered, the addressing advances to the next sub-bank (S), if applicable. Finally, once the entire row which is distributed across all sub-banks within all banks within both wings, such as the highlighted row in Figures 2.1 and 2.2, has been covered, does the row number (R) advance.

The RSBCW layout was chosen to optimize for unit stride accesses. Since the highest priority (lowest significant bit field) in the address decoding determines the wing (W), successive accesses alternate between the two wings and multiple unit stride streams can proceed simultaneously without interfering with one another. Since the next priority is to the column access (C), all of the columns within a DRAM page are used before advancing to the next one, potentially reducing the number of precharges required within the DRAM array and saving energy.

2.2 Conflicts

Two major types of conflicts discussed in the studies below are bank and sub-bank conflicts. For understanding the results of the studies, it is useful to review the basic rules of access to the memory system, what can cause conflicts, and how different types of conflicts can interact with one another.

The interface between each wing and the CPU consists of 4 64-bit data buses. For the default VIRAM configuration, on each cycle those buses can be organized as either 1 256-bit unit stride access to a single column, or 4 64-bit non-unit stride accesses spread across 4 different banks. This organization is the same for all of the configurations included in this study with 4 lanes. Note that since only one of the memory units is capable of generating 4 separate 64-bit accesses, the only way in which all 8 buses could be used on a single cycle is if one wing was processing a unit stride access stream and the other wing was processing a non-unit stride access stream in which all 4 of the addresses each mapped to different banks within that wing. The number of data buses and the size of a unit stride access grouping scales with the number of lanes. For instance, with 8 lanes, the interface for each wing consists of 8 64-bit data buses which can be organized as either 1 512-bit unit stride access to a single (larger) column or 8 64-bit non-unit stride accesses spread across 8 different banks. If the accesses from the memory units do not meet the required criteria, bank conflicts will result, and one or more of the accesses will have to wait until the next cycle. Conflicts for the data buses connecting the wings to the CPU and conflicts between the banks of the DRAM array are both counted as bank conflicts, since they are both resolved at the same stage within the Vector Memory Functional Unit (VMFU) pipeline, the Conflict Resolution stage. There is no separate statistic for wing conflicts.

In the general case, the specific behavior within a DRAM sub-bank is governed by a series of parameters and is too complex to describe in detail here. See [8] for complete details. For the default VIRAM configuration (and for all other configurations included in this study), its behavior can be summarized as follows. Every time an access causes a row miss, the affected sub-bank is busy for some amount of time before another access that causes a row miss (and hence a precharge within the DRAM array) can proceed. For loads, the sub-bank busy time is 4 cycles; for stores, it is 9 cycles. Accesses within the same sub-bank (either loads or stores) that address different columns within the same row (row hits) can proceed at the rate of one per cycle. Sub-bank conflicts are resolved within the VMFU pipeline at the Memory Controller stage. In the presence of either bank or sub-bank conflicts, priority is always given to addresses from the instruction appearing first in program order. See [14] and [8] for a more detailed discussion of the VMFU pipeline stages.

The cost of a bank conflict is therefore 1 cycle, and the cost of a sub-bank conflict is therefore up to 4 cycles for loads and up to 9 cycles for stores.

2.3 Examples

We will illustrate by examples. Each example considers two consecutive accesses in isolation, ignoring the behavior of any previous or subsequent accesses. Assume that the first access causes a row miss; the second access is a row hit if and only if it maps to the same row within the same bank (and sub-bank, if applicable) as the first access. For example, if the first access mapped to the highlighted row in the configuration shown in Figure 2.1, the second access would be a row hit if and only if it mapped to the highlighted row and was also in the same bank. If the first access mapped to the highlighted row in the configuration shown in Figure 2.2, the second access would be a row hit if and only if it mapped to the highlighted row and was also in both the same bank and the same sub-bank.

For these examples, we will look at consecutive accesses spaced apart from one another at successively further powers of 2 length strides. Note that bank and sub-bank conflicts can occur within a single non-unit stride access stream, or between two (either unit or non-unit) access streams. Bank and sub-bank conflicts cannot occur within a single unit stride access stream.

Suppose that the two consecutive accesses both map to the same row within the same bank, the block marked `0` on Figure 2.1. This will in general cause a bank conflict, and if the first access (load or store) is on cycle i, the second access (load or store) cannot proceed until cycle $i + 1$. The only exception to this is that if the accesses are close enough together in memory that they both map to the same 64-bit word within the same column. In this case, they are merged together and both proceed together on cycle i. Note that merging only occurs for accesses within a single instruction sent to the memory system on the same cycle. Merging never occurs between different instructions or across different cycles.

Suppose that the two accesses are farther apart in memory. For instance, one is in block `0` and the other is in block `1` (the same bank placement but in the opposing wing) or one is in block `0` and the other is in block `2` (the next bank in the same wing). Since both accesses are in different banks in both cases, there is no conflict, and they can both proceed together on cycle i.

Suppose that the two accesses are even farther apart in memory, for instance block `0` and block `16`. In this case, they map back to the same bank, causing a bank conflict. In this example, because the second access is a new row (row miss) within the same sub-bank as the first access, as shown in Figure 2.1, this additionally causes a sub-bank conflict. If the first access is a load, the two accesses can occur at cycles i and $i + 4$ respectively. If the first access is a store, the two accesses can occur at cycles i and $i + 9$ respectively. If the bank is divided into multiple sub-banks, however, as in Figure 2.2, the sub-bank conflicts are eliminated, leaving only the bank conflicts. The two accesses can occur at cycles i and $i + 1$ respectively, for either loads or stores.

Suppose that the two accesses are farther apart still, for instance block `0` and block `64`. Even with 4 sub-banks, as in Figure 2.2, they still map to the same sub-bank, and the sub-bank conflict is not eliminated. The timing of the accesses is at cycles i and $i + 4$ if the first access is a load, and cycles i and $i + 9$ if the first access is a store.

Depending on the type of access pattern seen by an application, sometimes the bottleneck will be bank conflicts, sometimes it will be sub-bank conflicts, and sometimes it may be a mixture of the two. If all of the memory conflicts that an application is experiencing are bank conflicts, adding more sub-banks will not help. It should also be noted that sometimes bank conflicts can mask sub-bank conflicts. In other words, an access stream might otherwise experience sub-bank conflicts if there were no bank conflicts, if the accesses to new rows within the same sub-bank are spaced closer together than the sub-bank busy time. However, if there are bank conflicts, it is possible that the stalls generated from those bank conflicts will sufficiently separate the relevant accesses in time such that row misses are separated by at least the sub-bank busy time and there are no longer any sub-bank conflicts.

The preceding discussion and figures represent the behavior for the default layout of RSBCW. Alternative layouts would have different organizations and different behaviors for various regions of strided accesses. The basic hierarchical configuration, however, as well as what defines bank and sub-bank conflicts, remain the same as data layout is varied.

See [8] and [14] for more details about the VIRAM memory system.

Section 3

Micro-benchmarks

This study uses micro-benchmarks to study the limits of VIRAM memory system performance as seen by multimedia applications. Figure 3.1 shows a set of representative image sizes. The images were assumed to consist of pixels of 8-bit values, with multiple color planes such as RGB or YUV stored as distinct (non-interleaved) arrays. Since VIRAM-1 does not provide an 8-bit virtual processor width, `vpw` was set to 16. Figure 3.2 shows the patterns used to access each image. These patterns include scanning each image horizontally, vertically, and in a blocked fashion with 8×8 blocks. Also, a sampling of 10000 random pixels within the image was performed. The horizontal, vertical, and random benchmarks respectively measure unit stride, strided, and indexed bandwidth. The blocked benchmark measures the bandwidth for an access pattern found in the DCT performed by JPEG and MPEG codecs [11][12][23][22][19]. There are two versions of each micro-benchmark, one to measure load bandwidth and one to measure store bandwidth. Each of the benchmarks were run for each of the image sizes, adjusting various software and hardware parameters as described in the sections that follow.

These micro-benchmarks are useful for exploring the limits of memory system performance as seen by multimedia applications. In a real application, there would be some amount of computation being performed within the inner loop, and not simply loads and/or stores. These benchmarks are still useful for determining what the best memory bandwidth is that can possibly be utilized for different types of accesses, much in the same way that lmbench [21] is used to characterize the memory performance of hardware systems.

Image Size	**Total Size**	**Aspect Ratio**	**Notes**
128 × 96	12,288	4:3	SQCIF
176 × 144	25,344	11:9	QCIF
352 × 240	84,480	22:15	SIF. CD WhiteBook Movies, video games.
352 × 288	101,376	11:9	CIF
352 × 480	168,960	11:15	HHR. VHS equivalent
480 × 480	230,400	1:1	Bandlimited (4.2 MHz) broadcast NTSC.
512 × 384	196,608	4:3	Macintosh.
544 × 480	261,120	17:15	Laserdisc, D-2, Bandlimited PAL/SECAM.
640 × 480	307,200	4:3	Square pixel NTSC. Advanced TV (SDTV). VESA (VGA Graphic, XGA).
704 × 480	337,920	22:15	Advanced TV (SDTV)
720 × 400	288,000	9:5	VGA Text.
720 × 480	345,600	3:2	CCIR 601. Studio D-1.
800 × 600	480,000	4:3	VESA (SVGA).
832 × 624	519,168	4:3	Macintosh.
1024 × 768	786,432	4:3	VESA (SVGA, XGA). Macintosh.
1152 × 864	995,328	4:3	Macintosh.
1280 × 720	921,600	16:9	Advanced TV (HDTV)
1280 × 1024	1,310,720	5:4	VESA (SVGA).
1600 × 1200	1,920,000	4:3	VESA (SVGA).
1800 × 1440	2,592,000	5:4	
1920 × 1080	2,073,600	16:9	HD-CIF
1920 × 1200	2,304,000	8:5	Advanced TV (HDTV)

Figure 3.1: Image Sizes. Aspect ratios listed refer only to the ratio of horizontal to vertical pixels. This ratio corresponds to the dimensions of the visual display if and only if the height and width of the individual pixels are equal; pixels can be rectangular in shape. Information compiled from [22], [5], [1], [19], [26], [7], [25], [10], [3], and [24].

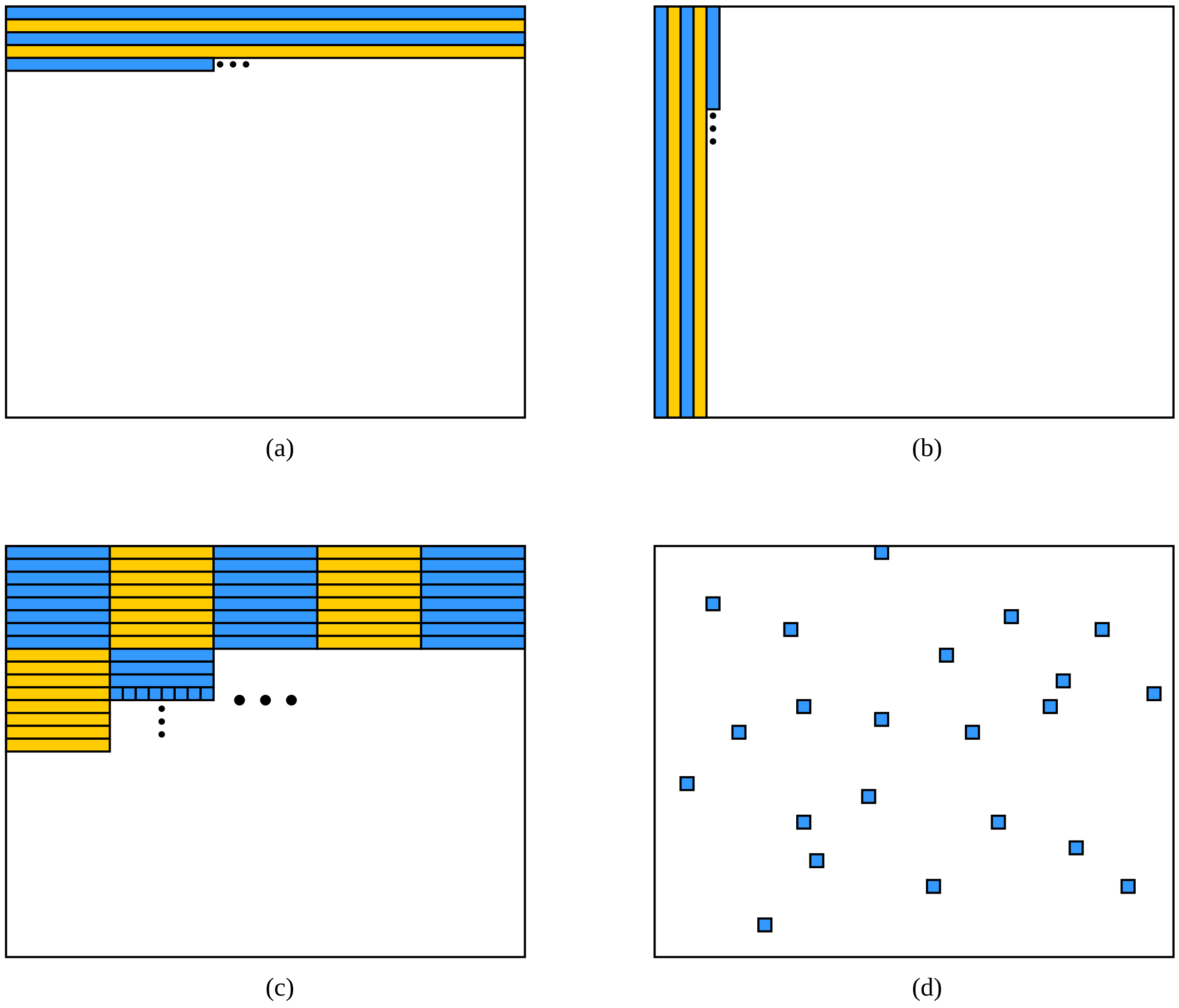


Figure 3.2: Image Access Patterns. It is assumed that pixels adjacent horizontally are stored adjacent in memory. The horizontal access pattern (a) is effectively one large unit stride, with a vector length equal to the image size. The vertical access pattern (b) is a series of strided accesses, each with a stride equal to the image width and a vector length equal to the image height. The blocked access pattern (c) has a series of 8 unit stride accesses each with a vector length of 8 to access one 8 $\times$ 8 pixel block. The blocks are then processed in a horizontally tiled pattern until the entire image is scanned. The random pattern (d) uses indexed operations to access 10000 pixels within the image. Alternating shadings are for illustrative purposes only. Images are not to scale.

Section 4

Metrics

The metric of interest for this study is the memory bandwidth achievable in each circumstance, and how that compares to peak. For VIRAM-1, the peak memory bandwidth provided by the hardware (2 memory units[1], 4 lanes, 64 bits per lane per cycle, at 200 MHz) is 12.8 GB/s. For these benchmarks, however, the effective peak is less. Figure 4.1 shows a summary of the peak bandwidths available under various conditions studied.

VIRAM-1 contains 4 physical lanes of 64 bits each. As the `vpw` is halved, the number of virtual lanes available doubles. The minimum `vpw` implemented in VIRAM-1 is 16 bits; there is no 8-bit `vpw`. Even though the pixels that form an image are 8-bit data, they have to be processed in VIRAM-1 using a 16-bit `vpw`.[2] Since only half of the 16-bit virtual lane is useful for transferring data to and from memory, the peak bandwidth available for unit stride accesses is halved to 6.4 GB/s.

Configuration	**Unit stride**	**Strided/indexed**
Default	*6.4 GB/s*	*0.8 GB/s*
1 lane	1.6 GB/s	0.2 GB/s
2 lanes	3.2 GB/s	0.4 GB/s
4 lanes	*6.4 GB/s*	*0.8 GB/s*
8 lanes	12.8 GB/s	1.6 GB/s
4 address generators	*6.4 GB/s*	*0.8 GB/s*
8 address generators	6.4 GB/s	1.6 GB/s
16 address generators	6.4 GB/s	3.2 GB/s

Figure 4.1: Peak Bandwidths. Peak bandwidths available for unit stride and strided/indexed loads and stores of 8-bit pixel data in a 16-bit `vpw` for VIRAM-1 and for several alternative configurations that scale the number of lanes or address generators. The "4 lanes" and "4 address generators" configurations, marked in *italics*, are both equivalent to the default.

[1]A recent VIRAM-1 design decision has been to implement only 1 memory unit. This study assumes the presence of 2 memory units. The second memory unit is only capable of processing unit stride instructions. The implications of dropping the second memory unit are discussed in Section 7.

[2]The reason for this decision is that it would be extremely unlikely for a real application to load and store data without performing some intervening computation, and the intermediate results of the computation are likely to require additional precision for accuracy. Therefore, when operating on 8-bit data, it is likely that one would have to set `vpw` to 16-bits, diminishing the usefulness of implementing an 8-bit `vpw`.

For strided and indexed accesses, the available bandwidth is reduced further. Only one of the memory units supports non-unit stride accesses, which would halve the peak again to 3.2 GB/s. However, at `vpw` = 16 bits, there are 16 virtual processors across the 4 lanes, meaning that this full 3.2 GB/s could only be achieved if 16 addresses were generated per cycle. But address generation resources are expensive, and VIRAM-1 can only generate 4 addresses per cycle across all of the lanes within one memory unit, regardless of the `vpw`. This limitation is present not simply due to the logic required to actually generate the addresses. Each virtual address needs to be translated to a physical address, requiring an additional read port for the TLB; each address needs to be checked against each other address to resolve conflicts, and the logic to do so grows as $O(n^2)$; and more addresses from the vector unit cause additional invalidation traffic to the scalar unit to keep the scalar caches consistent.[3] This address generation restriction further quarters the effective peak bandwidth for strided and index accesses, down to 0.8 GB/s.

The numbers listed above (6.4 GB/s and 0.8 GB/s) are for the unit stride and strided/indexed bandwidths respectively of the default VIRAM-1 configuration, with 4 lanes and 4 address generators per memory unit. Some of the studies that follow investigate scaling the number of lanes both down and up, to 1, 2, 4, and 8 lanes. The peak bandwidths scale with the number of lanes, resulting in peak unit stride bandwidths of 1.6, 3.2, 6.4, and 12.8 GB/s and peak strided/indexed bandwidths of 0.2, 0.4, 0.8, and 1.6 GB/s.

Some of the studies that follow investigate scaling up the number of address generation resources beyond the default of 4, to 8 and 16 address generators. At 16 address generators, each virtual processor (VP) is able to generate an address on every cycle for the minimum `vpw` of 16 bits. This limitation only affects strided/indexed operations, since unit stride accesses span all of the bits of a single element group with a single address that covers all of the lanes within one memory unit. For these cases (holding the number of lanes constant at 4), the peak non-unit stride bandwidths are 0.8, 1.6, and 3.2 GB/s. The unit stride peak bandwidth stays constant at 6.4 GB/s.

[3] Scalar cache invalidations are not yet modeled in `vsim-p`.

Section 5

Methodology

This study measures the memory bandwidth achieved within the inner loop of each micro-benchmark. Although this does include the loop overhead of the strip-mined loop, it does not include initial setup such as setting `vpw` nor the time for the vector pipelines to fill and drain. This simplification is justified because such costs would likely be borne by an application relatively infrequently – ideally just once per context switch, although realistically perhaps more often, maybe even once per function, depending on the implementation of memory barriers and the intelligence of the compiler in using as few barriers as possible. Either way, it is not useful to include such costs in a micro-benchmark. This methodology is best summarized by the observation that if both memory units are completely busy for duration of the memory accesses in a benchmark, the goal is to measure that as 100%.

To avoid measuring the time for the pipelines to drain, and to instead measure the throughput, the following methodology was used. We take a time snapshot (the start time) when the first relevant memory instruction is issued to a VMFU, by setting a breakpoint at the appropriate PC and redirecting `echo $cycle` [8] to an output file. This breakpoint is then deleted, and a breakpoint is set to measure when a carefully constructed, unrelated (dummy) memory instruction that immediately follows all of the relevant instructions is issued to either VMFU0 or VMFU1 (the end time). Whether the dummy instruction should be constructed so that it issues to VMFU0 or VMFU1 is dependent on the benchmark. The start time is subtracted from the end time, and by dividing by the usable bytes transferred (for example, the loading of the indices for indexed loads and stores are *not* counted) and the clock cycle time, the memory bandwidth is calculated. Future work includes adding simulator-specific vector NOP instructions to the simulation tools to replace the dummy instruction used here for timing purposes.

This methodology is reasonably accurate, although there are some instances in which minor measurement errors can occur. Instructions spread out both in time and space while they are executing; various elements of a single instruction can be at various points within a pipeline, and all of the elements do not necessarily advance at the same rate. For measurement purposes, some point has to be chosen at which to draw the line and count as the time for an instruction, and the point at which the instruction issues to some Vector Functional Unit (VFU) was used. As long as this point is held constant, the time differential between two instructions is reasonably accurate, but not always so. For instance, in one case of strided accesses, each instruction needs to stall at the address generation stage (which is after VFU issue) for 3 cycles due to insufficient address generation resources. The measured end time is short by 3 cycles since it does not account for the stall of the last relevant instruction. This artifact results in what ought to be 100% utilization being measured erroneously as 100.25%. Such errors are not present in all cases, and when present they are relatively minor.

Section 6

Studies

The four memory micro-benchmarks studied include horizontal, vertical, 8 × 8 blocked, and random image access patterns.

6.1 Horizontal Image Access Pattern

A horizontal access pattern for an image is effectively one large unit stride, with a vector length equal to the image size. This pattern is the easiest case for achieving peak performance.

Figure 6.1 shows the load and store bandwidths across all of the image sizes for optimized and unoptimized code, and as the image array is aligned and unaligned with respect to a 256-bit boundary. The unoptimized code is the initial, naive implementation. The optimized code unrolls the inner loop twice and utilizes branch delay slots.

For each case, the performance is almost constant regardless of image size. This consistency is because for all but short vectors, the performance of unit stride loads and stores will be relatively constant with vector length, and even the smallest image has over 10,000 pixels.

Since the memory accesses in this benchmark are entirely unit stride, both memory units can be utilized. However, without having the loop unrolled, there is not enough issue bandwidth to keep both units busy. There are 2 empty cycles before issuing each instruction to the vector unit. Given 8 element groups per instruction (at `vpw` = 16 bits and 4 lanes, it takes 8 cycles of using all 16 VPs each cycle to span the `mvl` of 128), using 8 out of 10 cycles would give 80% of peak performance. However, the actual performance achieved in this case is only 67%. To understand why, we need to examine how multiple memory access streams can interfere with each other in the presence of empty cycles or stalls.

With the default VIRAM memory layout of RSBCW, a unit stride access stream alternates between the two wings. (The only exception would be for a load or store that spanned no larger than a single 256-bit column access to a single wing.) This pattern can be seen by examining the order of the accesses shown in Figure 2.1. The reason for this pattern is that only a single unit stride access can go to each wing on a single cycle. Figure 6.2 (a) shows that if there are two access streams, they both alternate between opposing wings, and in the steady state they do not interfere with each other and there are no stalls. Figure 6.2 (b) shows that since the data being loaded and stored is 8 bits wide in these micro-benchmarks but the `vpw` is 16 bits, only half of the physical lane width (128 bits out of 256) is used each cycle and the pattern alternates at half

Code Optimized?	**No**	**No**	**Yes**	**Yes**
Data Aligned?	**No**	**Yes**	**No**	**Yes**
Peak Bandwidth	**6.4 GB/s**	**6.4 GB/s**	**6.4 GB/s**	**6.4 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (99%)
176 × 144	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
352 × 240	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
352 × 288	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
352 × 480	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
480 × 480	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
512 × 384	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
544 × 480	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
640 × 480	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
704 × 480	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
720 × 400	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
720 × 480	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
800 × 600	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
832 × 624	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1024 × 768	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1152 × 864	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1280 × 720	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1280 × 1024	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1600 × 1200	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1800 × 1440	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1920 × 1080	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1920 × 1200	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
Median	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
Mean	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
Standard deviation	0.00 (0%)	0.00 (0%)	0.00 (0%)	0.01 (0%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (99%)
176 × 144	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
352 × 240	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
352 × 288	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
352 × 480	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
480 × 480	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
512 × 384	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
544 × 480	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
640 × 480	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
704 × 480	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
720 × 400	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
720 × 480	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
800 × 600	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
832 × 624	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1024 × 768	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1152 × 864	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1280 × 720	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1280 × 1024	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1600 × 1200	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1800 × 1440	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1920 × 1080	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
1920 × 1200	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
Median	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
Mean	4.3 (67%)	4.3 (67%)	5.1 (80%)	6.4 (100%)
Standard deviation	0.00 (0%)	0.00 (0%)	0.00 (0%)	0.01 (0%)

Figure 6.1: Load and store bandwidth for horizontal image access pattern. Optimized code has the inner loop unrolled twice and utilizes branch delay slots. Unoptimized code is not unrolled and does not use branch delay slots. Alignment of data refers to 256-bit boundaries.

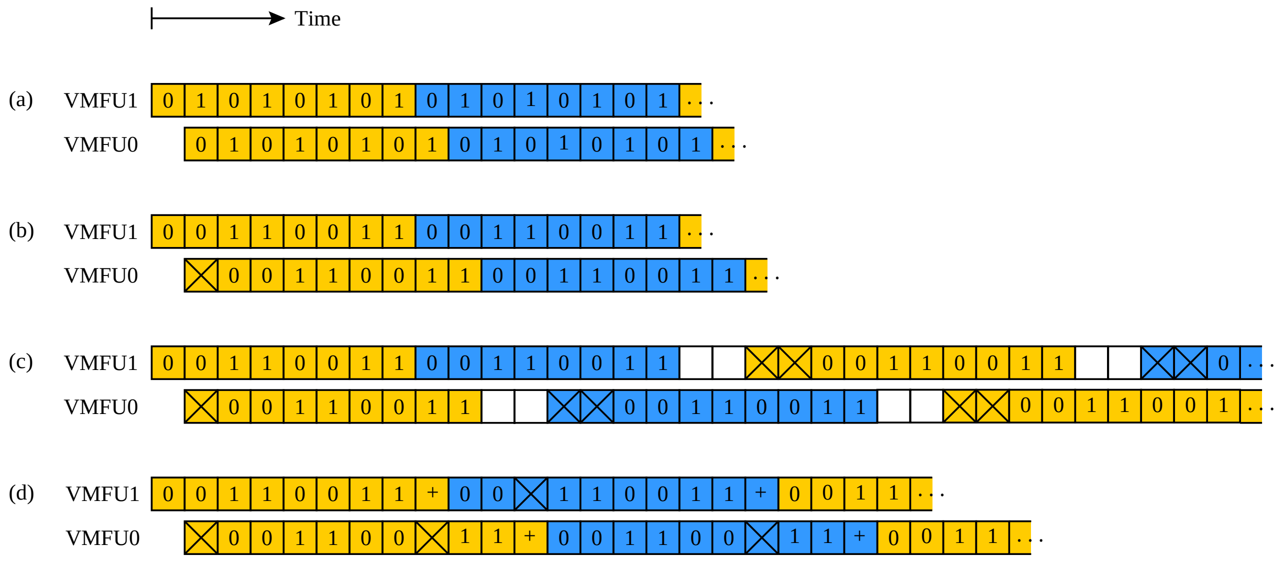


Figure 6.2: Alternating wing pattern from parallel unit stride access streams. Time advances to the right within each VMFU. "0" and "1" are used to indicate to which wing the address for a given element group belongs. For each case, at any point in time (a vertical slice), there can only be one access to each of the wings. Alternating shadings are used to mark divisions between instructions. Stall cycles from memory conflicts are marked with "×". Empty cycles from insufficient issue bandwidth are indicated with a blank box. Extra element groups at the end of an unaligned unit stride access stream are marked with a "+". (a) shows the typical VIRAM-1 behavior. (b) shows the behavior for 8-bit data in a 16-bit `vpw`. (c) shows 2 empty cycles from insufficient issue bandwidth causing 2 extra stall cycles, giving 67% of peak performance. (d) shows 1 extra cycle from an unaligned access causing 1 extra stall cycle, giving 80% of peak performance.

of this rate. Figure 6.2 (c) shows that if one memory unit is empty for 2 cycles, the alternating pattern is disturbed, and a stall of another 2 cycles is needed for it to correct itself. Therefore, with the loop not unrolled, 4 cycles are wasted for every 8 cycles of useful work, and 8 out of 12 gives 67% of peak performance.

Unaligned unit stride access streams cause the memory unit to be busy for an extra cycle at the end of each instruction. With the loop not unrolled, since there are already empty cycles, this extra busy cycles does not hurt performance further, and the performance is still 67% of peak.

The particular problem illustrated in Figure 6.2 (c), that of insufficient issue bandwidth from not unrolling the loop, is somewhat of a simulation artifact. The scalar core to be used in VIRAM-1 is capable of fetching 2 instructions on every cycle and sending 2 instructions per cycle across the coprocessor interface to the vector unit. The current, simplified model of the core within `vsim-p` is single-issue, and it only sends 1 instruction per cycle to the vector unit. This case was worked out by hand for a dual-issue scalar core, and the result is that there is then sufficient issue bandwidth to keep both memory units busy. The 2 empty cycles go away, as do the subsequent stall cycles. The behavior therefore resembles that shown in Figure 6.2 (b), which gives 100% of peak performance. For a dual-issue scalar core, therefore, we would expect the first and second results columns of Figure 6.1 to resemble the third and fourth columns respectively. This simulation artifact is still a useful result, however, in that it highlights the importance of scalar issue bandwidth for keeping the vector units busy, as well as how empty cycles from insufficient issue bandwidth can lead to further stalls in the VIRAM memory system. Future work includes integrating a more detailed, dual-issue model of the scalar core into the VIRAM simulator.

Once the code is optimized by utilizing the branch delay slots and unrolling the loop twice, there is now sufficient issue bandwidth to keep both memory units busy. If the image array is 256-bit aligned, then each unit stride access is aligned, and 100% of peak performance is achieved. If the accesses are unaligned, this costs one extra element group for every 8. Performing 8 cycles of useful work out of every 9 would give 89% of peak performance. Similar to the case outlined above, however, the loss is greater than that. Figure 6.2 (d) shows that the extra element group causes a stall for 1 cycle to maintain the alternating access pattern. Using 8 cycles out of every 10 gives 80% of peak performance.

Figure 6.3 shows how well this benchmark scales with respect to the number of lanes, as the array is kept aligned (to a boundary of 64 bits times the number of lanes), and the code is kept optimized (using branch delay slots and unrolled twice).

The horizontal access benchmark scales down perfectly; 1, 2, and 4 lanes give 100% of peak performance. As the number of lanes is scaled up, however, the vector length relative to the hardware has decreased, and there is once again a problem of not having enough instruction issue bandwidth to keep both memory units busy. The problem is made worse due to the resulting empty cycles interfering with the access pattern that alternates between the wings, as was true in the examples above. The net result is that there are only 4 element groups per instruction (at 8 lanes), but for every instruction there are 3 wasted cycles. Using 4 out of every 7 cycles gives 57% of peak performance.[1]

The issue bandwidth problem can be addressed for 8 lanes by further unrolling the loop, to 4 times. However, the bandwidth achieved is still not 100%, and it is not issue bandwidth that is to blame. Unrolling again to 8 times (not shown in the figure) does not help. The problem in this instance is combination of the effects of simplified pipeline control and short vectors. Figure 6.4 (a) shows the effect of the simplified pipeline control in VIRAM-1; a stall at some point in any VFU causes all element groups from all instructions later in program order in all VFUs that are at the same or earlier pipeline stage to stall. This effect can have further consequences due to the interactions between the two memory access streams. The stall in one memory VFU disturbs the ping-ponging effect between the wings; Figure 6.4 (b) shows that a few cycles later there is another stall to realign the accesses. This stall again affects other instructions later in program earlier at earlier pipeline stages. Figure 6.4 (c) shows the net effect of 1 empty cycle for every 4 element groups; 4 out of 5 cycles utilized gives 80% of peak performance.

Both of the problems illustrated here, insufficient issue bandwidth, and stalls on one instruction affecting other instructions, become more of an issue as chimes get shorter and successive instructions are compressed closer together. Chimes are shorter with either shorter vector lengths or with more lanes. This is one potential difficulty to scaling performance in VIRAM as the number of lanes is scaled up.

6.2 Strided Access Patterns

A vertical access pattern for an image is a series of strided accesses. Before studying the performance of such an access pattern, it is useful to examine the properties of strided access for strides that are powers of 2, as this will motivate some potential optimizations.

[1]The performance in this case, the fourth result column in Figure 6.3, as for the previous cases of insufficient instruction issue bandwidth, would likely improve with a dual-issue scalar core.

Lanes	**1**	**2**	**4**	**8**	**8**
Loop Unrolled	**2**	**2**	**2**	**2**	**4**
Peak Bandwidth	**1.6 GB/s**	**3.2 GB/s**	**6.4 GB/s**	**12.8 GB/s**	**12.8 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**				
128 × 96	1.6 (100%)	3.2 (100%)	6.4 (99%)	7.3 (57%)	10.2 (80%)
176 × 144	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
352 × 240	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
352 × 288	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
352 × 480	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
480 × 480	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
512 × 384	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
544 × 480	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
640 × 480	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
704 × 480	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
720 × 400	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
720 × 480	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
800 × 600	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
832 × 624	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1024 × 768	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1152 × 864	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1280 × 720	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1280 × 1024	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1600 × 1200	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1800 × 1440	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1920 × 1080	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1920 × 1200	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
Median	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
Mean	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
Standard deviation	0.00 (0%)	0.00 (0%)	0.01 (0%)	0.00 (0%)	0.01 (0%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**				
128 × 96	1.6 (100%)	3.2 (100%)	6.4 (99%)	7.3 (57%)	10.2 (80%)
176 × 144	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
352 × 240	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
352 × 288	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
352 × 480	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
480 × 480	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
512 × 384	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
544 × 480	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
640 × 480	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
704 × 480	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
720 × 400	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
720 × 480	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
800 × 600	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
832 × 624	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1024 × 768	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1152 × 864	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1280 × 720	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1280 × 1024	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1600 × 1200	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1800 × 1440	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1920 × 1080	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
1920 × 1200	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
Median	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
Mean	1.6 (100%)	3.2 (100%)	6.4 (100%)	7.3 (57%)	10.2 (80%)
Standard deviation	0.00 (0%)	0.00 (0%)	0.01 (0%)	0.00 (0%)	0.01 (0%)

Figure 6.3: Load and store bandwidth for horizontal image access pattern. Data is aligned to the physical width in bits across all of the lanes. Number of lanes is varied.

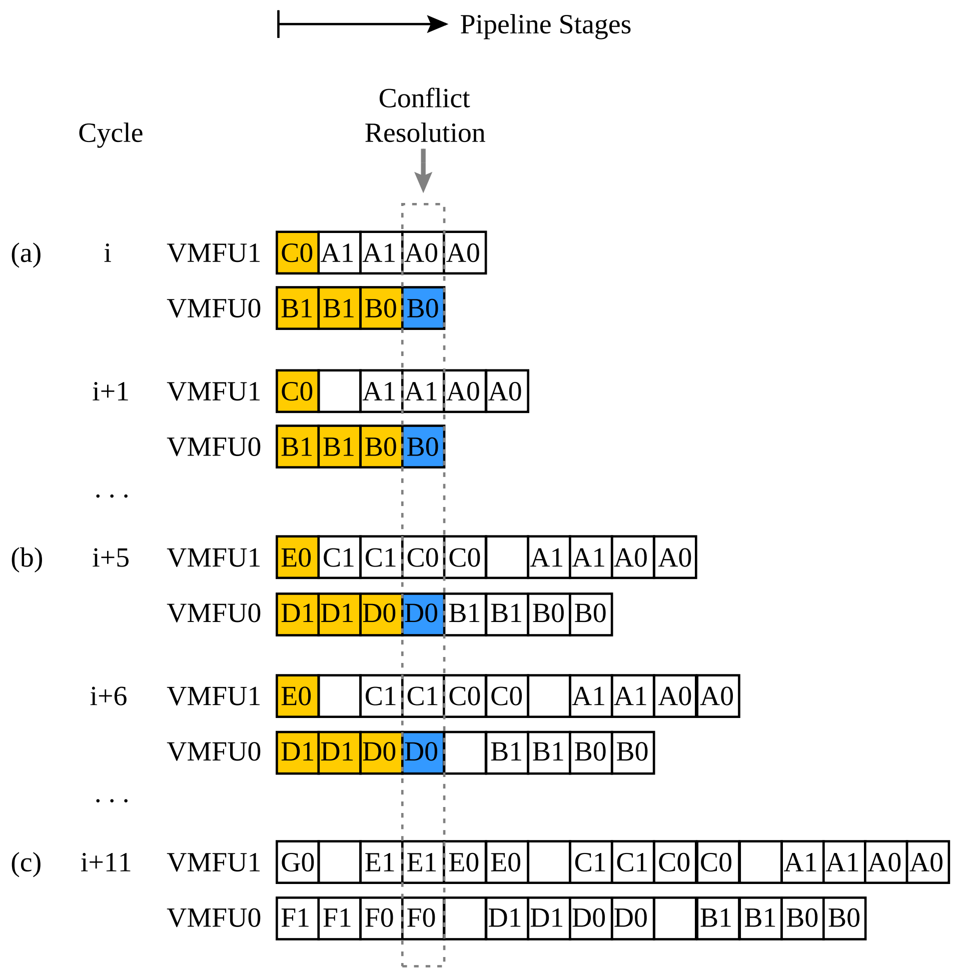

Pipeline Stages
Conflict Resolution
Cycle
(a) i
VMFU1 C0 A1 A1 A0 A0
VMFU0 B1 B1 B0 B0
i+1
VMFU1 C0 A1 A1 A0 A0
VMFU0 B1 B1 B0 B0
. . .
(b) i+5
VMFU1 E0 C1 C1 C0 C0 A1 A1 A0 A0
VMFU0 D1 D1 D0 D0 B1 B1 B0 B0
i+6
VMFU1 E0 C1 C1 C0 C0 A1 A1 A0 A0
VMFU0 D1 D1 D0 D0 B1 B1 B0 B0
. . .
(c) i+11
VMFU1 G0 E1 E1 E0 E0 C1 C1 C0 C0 A1 A1 A0 A0
VMFU0 F1 F1 F0 F0 D1 D1 D0 D0 B1 B1 B0 B0

Figure 6.4: Effects of simplified pipeline control and short vectors. Element groups move through the VMFUs from left to right; each vertical slice represents a pipe stage. Conflicts between the wings are resolved at pipe stage 4. Time advances downward. Each element group is represented by a letter and a number. The ordered sequence of letters (A, B, . . . , G) corresponds to the sequence of instructions in program order. "0" and "1" are used to indicate to which wing the address for a given element group belongs. A dark shaded element group indicates a stall. The stall affects all earlier element groups that are shaded light. Empty pipe stages resulting from stalls are indicated with blank boxes. Note that for clarity this notation is different from that used in Figure 6.2. With more lanes, the chimes are shorter, and short vectors can become a problem. A stall on one instruction is likely to affect other instructions as well. In (a), a stall at the leading element group of instruction B stalls the rest of B in the same VMFU, as well as instruction C in the other VMFU. In (b), a stall at the leading element group of instruction D stalls the rest of D in the same VMFU, as well as instruction E in the other VMFU. The cumulative effect of the stalls in this example, shown in (c), gives 80% of peak performance.

6.2.1 Power of 2 Length Strides

Figure 6.5 shows the performance for a series of 4096 strided accesses of 8-bit values through a region of memory large enough to hold all of the accesses, at various power of 2 length strides. The vertical axis is truncated to show the performance for non-unit stride accesses, which have a peak bandwidth of 0.8 GB/s for both loads and stores. (The unit stride bandwidth is off the scale at the peak of 6.4 GB/s). Strides were examined up to a value at which the stride size was equal to the square root of the memory size (32 MB), as this was deemed the largest stride to be of practical interest. A stride equal to the square root of the memory size could arise by accessing a column of a large square array spanning the entire memory stored in row major order. These figures are for the default memory layout of RSBCW. The default VIRAM-1 configuration is represented by the line labeled "0 XOR Levels".

The RSBCW layout is chosen to optimize for unit stride. The wing (W) has the highest priority (lowest significant bits in the decoding), so that multiple unit stride streams ping-pong between the two wings and do not interfere with one another, as described earlier. The next priority goes to the column access (C). The idea is that we want to re-use all of the columns within a DRAM page before advancing to the next. For power of 2 length strides, this layout gives several distinct regions of behavior described below, and evident in Figure 6.5. (The specific divisions between the regions refer to the behavior for 0 XOR levels.)

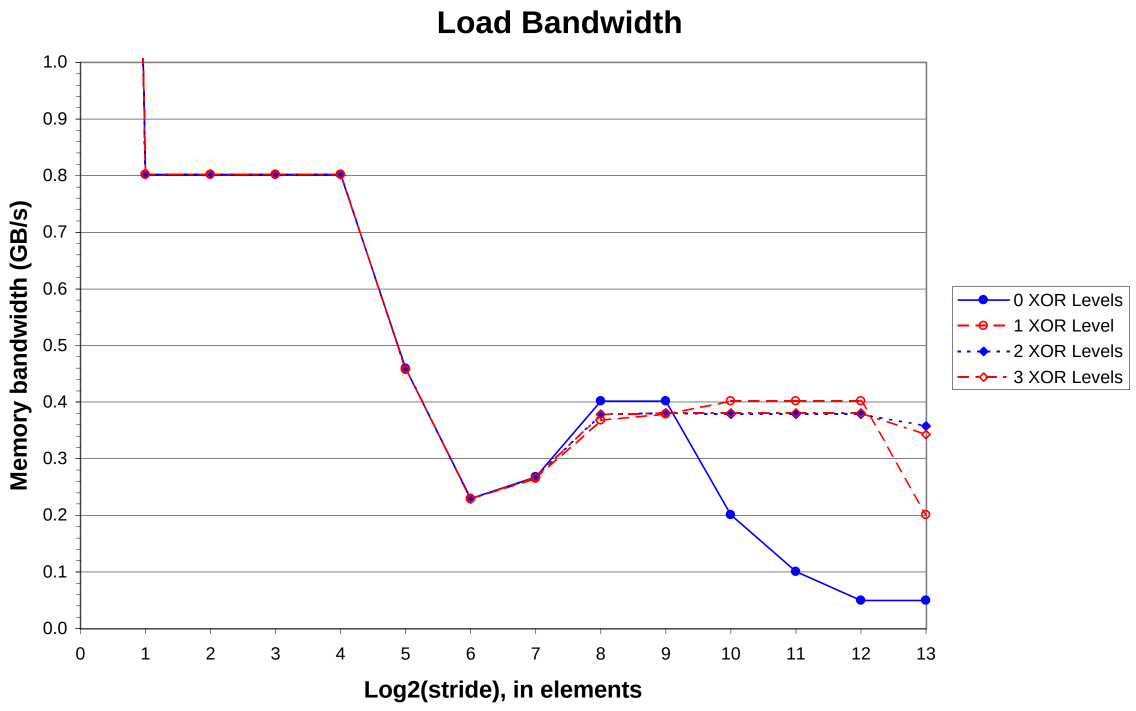
On each cycle there can be up to 4 8-bit addresses generated. For very small strides (<= 16) there are no conflicts and peak performance (0.8 GB/s) is achieved. At stride 2, the addresses span only 64 bits, the smallest unit of granularity for addressing the memory system, so they all merge together as if they were a single address. At strides 4 and 8, the addresses span 128 and 256 bits respectively. These all map to the same 256-bit column, so there are no conflicts. At stride 16, the addresses span 512 bits, which covers two columns. However, since those columns are located in opposing wings, both accesses can proceed simultaneously and there are still no conflicts.

As strides start to get bigger, the RSBCW layout becomes a problem. Multiple addresses within the same element group map to the same DRAM page, but different columns, causing bank conflicts. This effect is most pronounced at a stride of 64 elements, which equals 512 bits, where each successive access maps to the next column within the same DRAM page, resulting in a bandwidth of only 0.23 GB/s (29%), for both loads and stores.

As strides get bigger yet, the RSBCW layout is for a while no longer a problem; consecutive addresses are located in different banks, and there are no bank conflicts. At this point, however, we are limited by the fact that there are not multiple sub-banks per bank, and the sub-bank busy time means that we quickly cycle back to the same sub-bank and have to wait for the sub-bank to be free before another access can proceed. These sub-bank conflicts are more of a problem for stores than for loads, since the sub-bank busy time is longer. The best performance in this region is at a stride of 256 elements – 0.40 GB/s (50%) for loads, and 0.32 GB/s (40%) for stores.

At very large strides (>= 4096), consecutive addresses have cycled all the way around all of the banks and map to a different row in the same bank, producing bank (and sub-bank) conflicts on every access. This causes the performance to drop off severely, leveling out at 0.05 GB/s (6%) for loads and 0.02 GB/s (3%) for stores.

There are two possible alternatives for VIRAM-1 that could potentially improve the performance of strided loads and stores for power of 2 length strides – a different layout than RSBCW, and/or some hashing address interleaving scheme. A simple hashing scheme investigated here is adding one or more levels of XORing between the bank selection bits and the next most significant bits from the physical address. For instance, in the default layout shown in Figure 2.1, the bank is selected from the physical address by simply decoding the bits PA[11:9]. With 2 levels of XORing, the bank would be selected using PA[11:9] $\oplus$ PA[14:12] $\oplus$ PA[18:15].



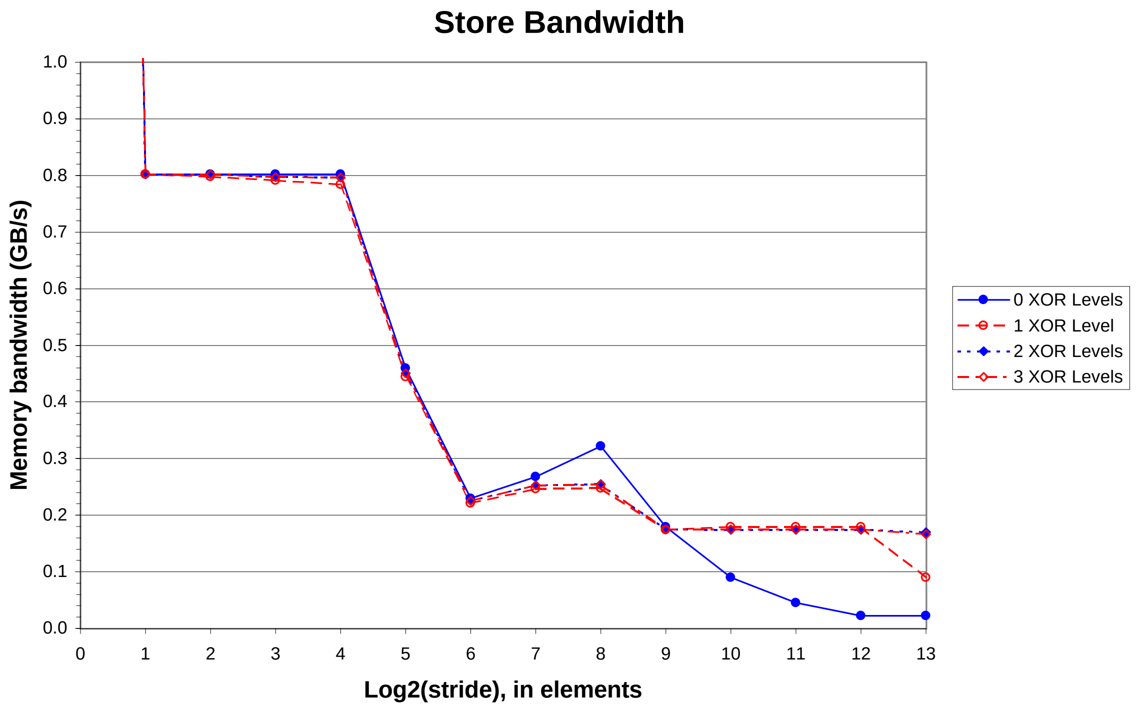


Figure 6.5: Load and store bandwidth for power of 2 length strides. Layout is fixed at RSBCW and XOR levels are varied. Peak bandwidth for non-unit strides is 0.8 GB/s. Vertical axis is truncated to show non-unit stride performance; unit stride bandwidth of 6.4 GB/s is off the scale.

As described further in [8], XORing the bank selection bits has the effect of somewhat scrambling the memory access pattern across the banks. (The pattern is still a repetitive one. XORing does not really randomize the pattern; it simply increases its periodicity.) Having multiple sub-banks within each memory bank would also help for certain situations, but that is not a viable alternative for VIRAM-1 due to limitations of the DRAM macro we are receiving from an industrial partner.

Figure 6.6 shows the results of using a layout of RCSBW instead of RSBCW, which means that the addresses cycle through the banks faster. This eliminates the dip down at stride 64 from consecutive addresses mapping to the same DRAM page. For loads, peak performance is able to be achieved at strides as big as 128 with RCSBW, versus 16 with RSBCW. However, since the banks are now being cycled through faster, it has just shifted the starting point of another problem. The boundary at which performance begins to drop for very large strides from having cycled around all of the banks for the RSBCW layout, shown in Figure 6.6, occurs at smaller strides (beyond 128 for loads and 16 for stores) than for the default layout (beyond 512 for loads and 256 for stores), shown in Figure 6.5.

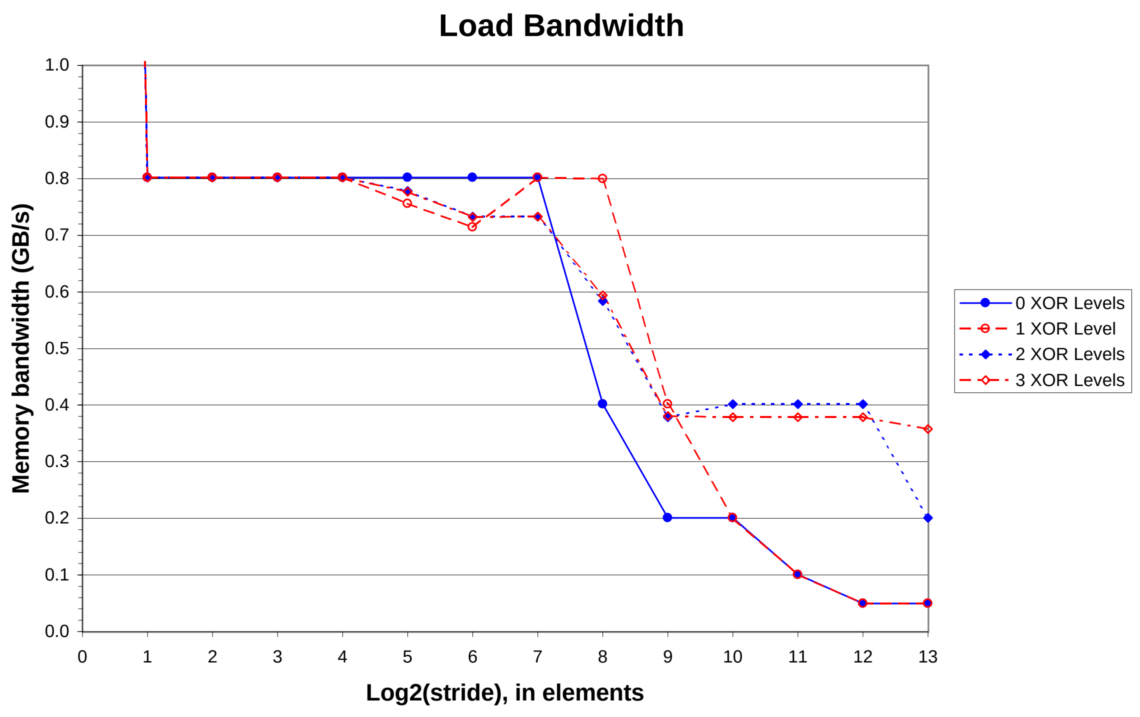
The problem associated with large power of 2 length strides, which occurs for both the RSBCW and RCSBW layouts but is more serious for RCSBW, can be addressed with the addition of levels of XORing of bank selection bits. As shown in Figures 6.5 and 6.6, successive levels of XORing push the drop off further out to larger strides for both layouts examined. For the default layout of RSBCW, two levels of XORing for loads and three levels for stores are sufficient to push the performance drop beyond the range of strides investigated.

The RCSBW layout does not affect the performance of a pure unit stride access stream like the horizontal image access pattern; it is still relatively easy to maintain 100% of peak bandwidth for the 4-lane case (not shown in any figure). Since RCSBW cycles through the banks faster than RSBCW, though, it could potentially consume more energy. This would be true if the length of the unit stride access was long enough to cycle through more than one bank with RCSBW, but short enough to not cycle through all banks with RSBCW. It is difficult to predict the performance impact of RCSBW (instead of RSBCW) on a unit stride access stream coexisting with a non-unit stride stream. Whether spreading out the unit stride accesses more would increase or decrease performance is dependent on the particular non-unit stride accesses with which conflicts might arise.

Based on this initial evidence, one might be tempted to conclude that a layout of RCSBW and the addition of XORing on the bank address bits is a good idea for strided accesses. It is important to recognize, however, that the specifics of the memory bandwidth achievable is highly dependent on the chosen stride. What is beneficial for some strides may be detrimental for others, and the above analysis only investigated power of 2 length strides.

Figure 6.7 shows the measured memory bandwidth for all strides within the region studied, comparing the layouts RSBCW to RCSBW, both at 0 XOR levels. Figure 6.8 keeps the layout constant at RSBCW and compares 0 and 2 XOR levels. (Note the switch from a log scale in the previous figures to a linear scale.) Based on these figures, the tentative conclusions above become much less certain. On average, the alternative layout (RCSBW) performs better than the default layout (RSBCW). However, there is a greater variance with the alternative layout, and there are clearly specific points at which the alternative layout performs much worse. For instance, there appear from the graph to be periodic local minima for RCSBW around every 1024th stride, starting at 512, that roughly coincide with periodic local maxima for RSBCW around similar strides (512, 1536, 2560, . . . , 7680). Looking at the raw data in detail, it can be seen the RCSBW minima are almost always (with one exception) exactly on the listed strides. For some as yet unexplained reason, the RSBCW maxima are always precisely 4 strides after the listed strides.

The addition of XOR levels behaves somewhat contrary to the alternative layout. Although the variance is decreased when adding XOR levels, so is the average bandwidth. There are some strides where adding



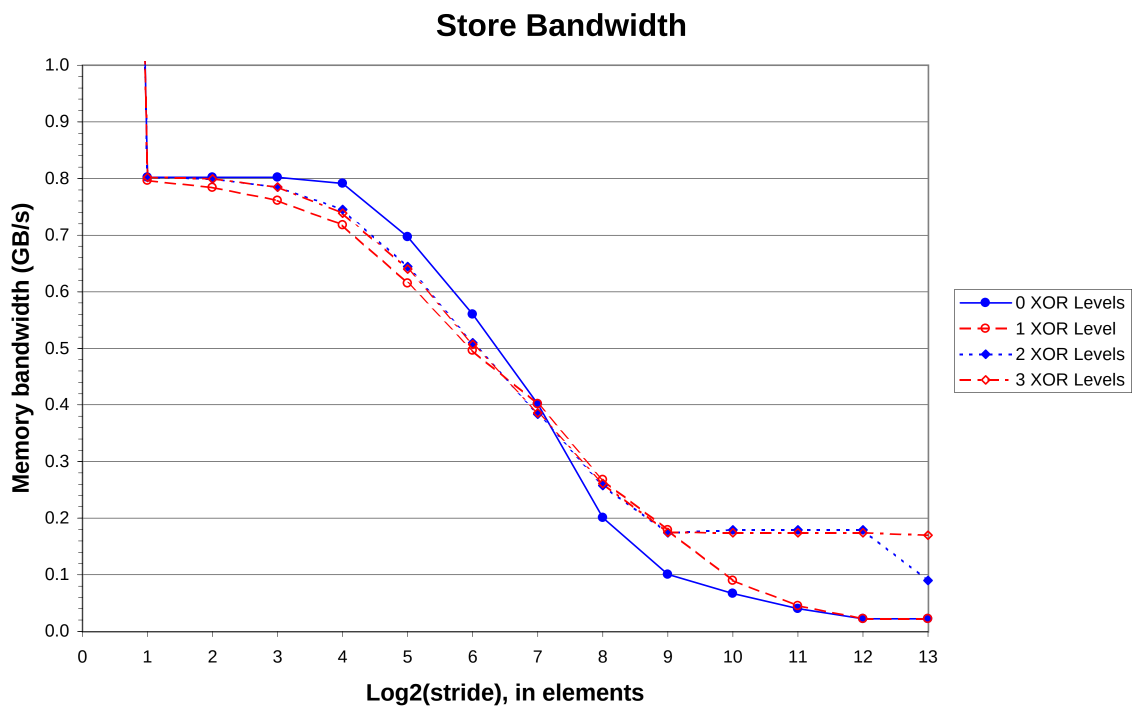
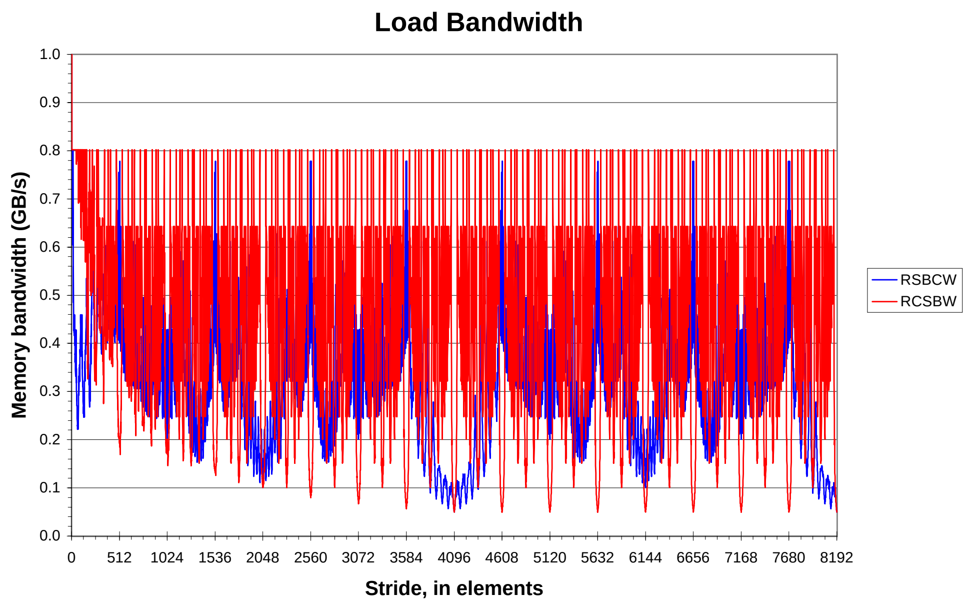
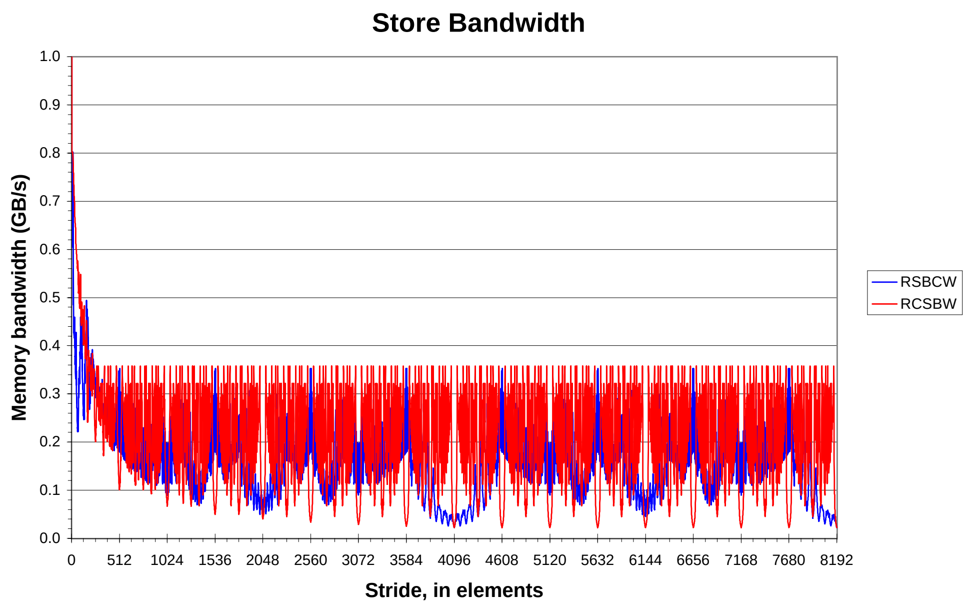


Figure 6.6: Load and store bandwidth for power of 2 length strides. Layout is fixed at RCSBW and XOR levels are varied. Peak bandwidth for non-unit strides is 0.8 GB/s. Vertical axis is truncated to show non-unit stride performance; unit stride bandwidth of 6.4 GB/s is off the scale.





	Bandwidth in GB/s (Percentage of Peak)			
	Load		**Store**	
Layout **Peak Bandwidth**	**RSBCW** **0.8 GB/s**	**RCSBW** **0.8 GB/s**	**RSBCW** **0.8 GB/s**	**RCSBW** **0.8 GB/s**
Median	0.32 (40%)	0.43 (54%)	0.16 (20%)	0.21 (27%)
Mean	0.31 (39%)	0.41 (51%)	0.16 (20%)	0.20 (25%)
Standard deviation	0.12 (15%)	0.19 (23%)	0.07 (9%)	0.10 (13%)

Figure 6.7: Load and store bandwidth for all strides. XOR levels are fixed at 0. Data layout is varied. Peak bandwidth for non-unit strides is 0.8 GB/s. Averages are for non-unit stride points only. Vertical axis is truncated to show non-unit stride performance; unit stride bandwidth of 6.4 GB/s is off the scale. *Since the RSBCW lines are somewhat hard to read, as they are partially obscured by the RCSBW lines, it should be noted that the RSBCW lines are the same as the corresponding "0 XOR Levels" lines in Figure 6.8.*

XOR levels consistently (and sometimes dramatically) improves performance, namely all sufficiently large powers of 2 (>= 1024). However, there are also some specific strides where adding XOR levels significantly decreases performance. If we again examine what is happening around every 1024th stride, starting at 512, switching from 0 XOR levels to 2 does not produce the same dramatic change (local minima instead of local maxima) that was seen above (RCSBW versus RSBCW), but the performance with 2 XOR levels is consistently significantly less than the local maxima found with 0 XOR levels.

The best generalization that can be made from these results is that it is very difficult to make generalizations. Averages can be somewhat misleading, since the behavior for a particular stride between two alternatives may be very different than the average. Although this discussion was useful to provide the context for why we may wish to consider an alternative layout and/or XORing of bank selection bits, what matters most is the behavior of the specific strides of concern, those seen by a vertical access pattern for a representative set of image sizes.

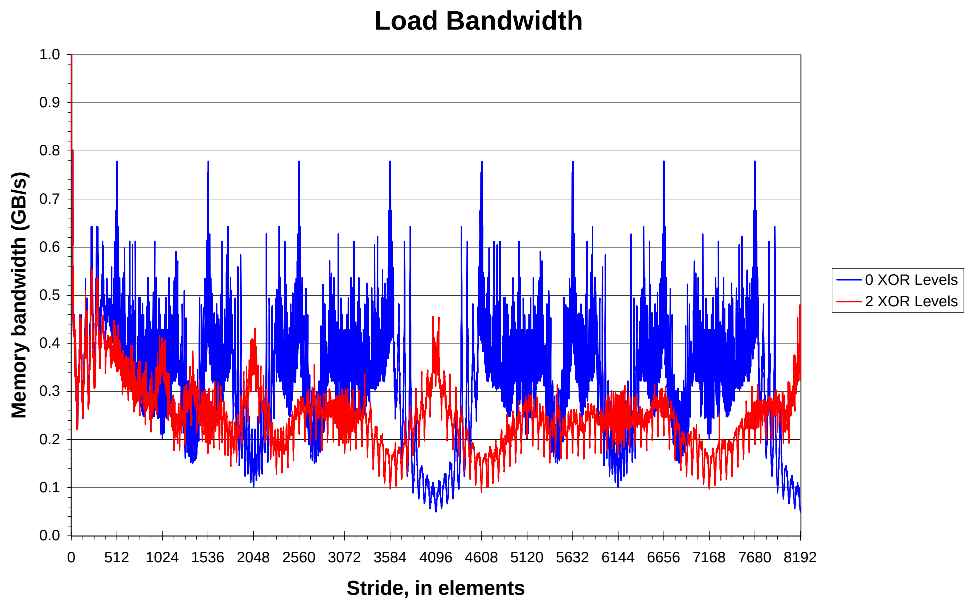
6.2.2 Vertical Image Access Pattern

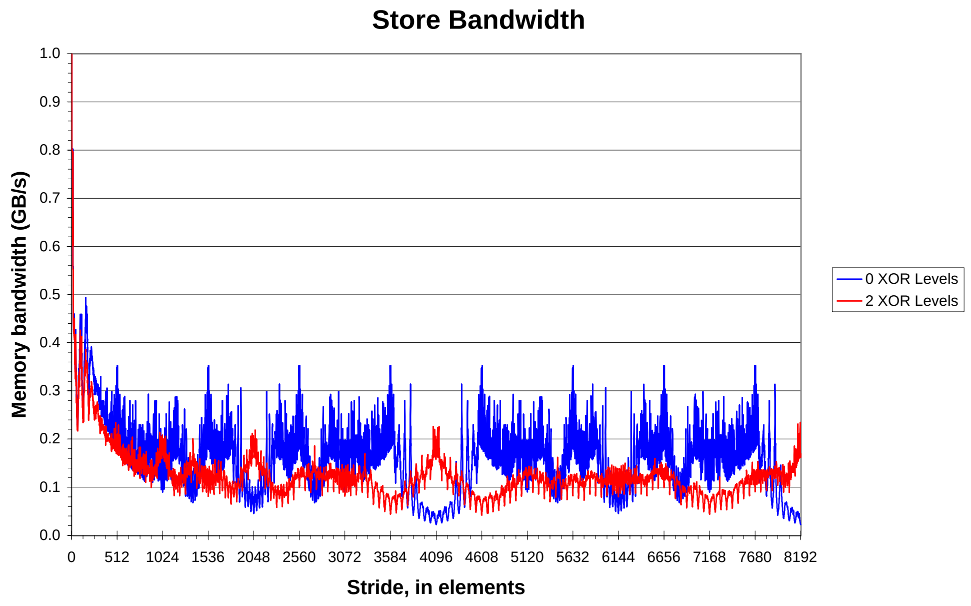
For the horizontal image access pattern, which consists of unit stride loads and stores, the benefits of aligning data and optimizing the code by unrolling the inner loop were examined. The vertical image access pattern consists of a series of strided accesses, each with a stride equal to the image width and a vector length equal to the image height. Because unit stride accesses group the 4 64-bit buses into a single 256-bit access, data alignment was a potential factor for the horizontal image access pattern. Since non-unit stride accesses individually address each 64-bit data bus, data alignment is not an issue for the vertical accesses. There is also no need to bother unrolling the inner loop. Loop unrolling was needed for the horizontal access pattern to best take advantage of both memory units. With the strided loads and stores of a vertical image pattern, only one memory unit can be utilized, and the performance is far enough away from the peak that instruction issue bandwidth is not an issue and there is no need to unroll the inner loop.

Figure 6.9 shows the effects of adding XOR levels to the bank selection bits for loads and stores for the default layout of RSBCW. Figure 6.10 shows the same effects for the alternative layout of RCSBW. The overall behavior from adding XOR levels for the strides used for vertically accessing images is similar to what was found when all strides were examined in the previous section. For both layouts, for both loads and stores, there are a few specific cases (most notably when the image width is a power of 2) in which levels of XORing improve performance, and the presence of XORing does reduce the variance. However, on average, the performance is worse with XORing than without. One partial exception is that for the RCSBW layout, the highest median bandwidth for both loads and stores is achieved with 1 XOR level; the mean still shows the highest bandwidth at 0 XOR levels.

The finding that the addition of XORing to the bank selection bits decreases the average bandwidth, as seen both when all strides within a range were considered (Figure 6.8) and for the vertical image access patterns (Figures 6.9 and 6.10) was somewhat unexpected. This simple address hashing scheme was considered because it is trivial to implement, requiring only a few gates, and because it addresses a specific known problem, that of large power of 2 length strides. Performance is indeed improved tremendously for those strides. However, there is a tradeoff. Most strides are not large powers of 2, including most of the image widths studied here, and for many of those strides the addition of XORing decreases performance. It is important to keep in mind that XORing is only one of many possible hashing schemes. Future work includes investigating other, possibly more complex, hashing schemes that address specific problem cases without decreasing the average bandwidth.

The effects of the alternative layout (RCSBW versus RSBCW) can be seen be comparing each of the points on Figure 6.10 with the corresponding points on Figure 6.9. There are some cases in which the alternative layout helps performance considerably; in other cases it hurts performance. On average, the alternative





	Bandwidth in GB/s (Percentage of Peak)			
	Load		**Store**	
XOR Levels	**0**	**2**	**0**	**2**
Peak Bandwidth	**0.8 GB/s**	**0.8 GB/s**	**0.8 GB/s**	**0.8 GB/s**
Median	0.32 (40%)	0.25 (31%)	0.16 (20%)	0.12 (15%)
Mean	0.31 (39%)	0.25 (31%)	0.16 (20%)	0.13 (16%)
Standard deviation	0.12 (15%)	0.07 (9%)	0.07 (9%)	0.06 (7%)

Figure 6.8: Load and store bandwidth for all strides. Layout is fixed at RSBCW. XOR levels are varied. Peak bandwidth for non-unit strides is 0.8 GB/s. Averages are for non-unit stride points only. Vertical axis is truncated to show non-unit stride performance; unit stride bandwidth of 6.4 GB/s is off the scale.

XOR Levels **Peak Bandwidth**	**0** **0.8 GB/s**	**1** **0.8 GB/s**	**2** **0.8 GB/s**	**3** **0.8 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.26 (33%)	0.27 (33%)	0.27 (33%)	0.27 (33%)
176 × 144	0.41 (51%)	0.40 (50%)	0.41 (51%)	0.41 (51%)
352 × 240	0.60 (75%)	0.43 (54%)	0.45 (56%)	0.45 (56%)
352 × 288	0.60 (75%)	0.43 (54%)	0.44 (55%)	0.44 (55%)
352 × 480	0.60 (75%)	0.43 (54%)	0.44 (55%)	0.44 (55%)
480 × 480	0.53 (66%)	0.47 (58%)	0.44 (55%)	0.45 (56%)
512 × 384	0.40 (50%)	0.33 (41%)	0.33 (41%)	0.33 (41%)
544 × 480	0.53 (66%)	0.43 (53%)	0.41 (51%)	0.41 (51%)
640 × 480	0.32 (40%)	0.27 (34%)	0.27 (34%)	0.27 (34%)
704 × 480	0.31 (38%)	0.25 (31%)	0.26 (32%)	0.26 (32%)
720 × 400	0.55 (69%)	0.31 (39%)	0.30 (38%)	0.30 (38%)
720 × 480	0.55 (69%)	0.31 (39%)	0.31 (38%)	0.30 (38%)
800 × 600	0.48 (60%)	0.30 (38%)	0.31 (39%)	0.31 (39%)
832 × 624	0.25 (31%)	0.23 (29%)	0.23 (29%)	0.23 (29%)
1024 × 768	0.20 (25%)	0.37 (46%)	0.35 (44%)	0.35 (44%)
1152 × 864	0.34 (42%)	0.18 (23%)	0.18 (23%)	0.18 (23%)
1280 × 720	0.18 (22%)	0.18 (22%)	0.18 (23%)	0.18 (23%)
1280 × 1024	0.18 (22%)	0.18 (22%)	0.18 (23%)	0.18 (23%)
1600 × 1200	0.32 (40%)	0.18 (22%)	0.18 (23%)	0.18 (22%)
1800 × 1440	0.39 (48%)	0.21 (27%)	0.21 (27%)	0.21 (27%)
1920 × 1080	0.17 (21%)	0.21 (26%)	0.19 (24%)	0.20 (25%)
1920 × 1200	0.17 (21%)	0.21 (26%)	0.19 (24%)	0.20 (25%)
Median	0.36 (45%)	0.29 (36%)	0.29 (36%)	0.29 (36%)
Mean	0.38 (47%)	0.30 (37%)	0.30 (37%)	0.30 (37%)
Standard deviation	0.16 (19%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.24 (30%)	0.26 (33%)	0.26 (33%)	0.26 (33%)
176 × 144	0.36 (46%)	0.33 (41%)	0.33 (42%)	0.33 (42%)
352 × 240	0.27 (34%)	0.22 (28%)	0.23 (28%)	0.23 (28%)
352 × 288	0.27 (34%)	0.22 (28%)	0.23 (28%)	0.23 (28%)
352 × 480	0.27 (34%)	0.22 (28%)	0.23 (28%)	0.23 (28%)
480 × 480	0.28 (36%)	0.23 (29%)	0.22 (28%)	0.22 (28%)
512 × 384	0.18 (22%)	0.16 (20%)	0.16 (20%)	0.16 (20%)
544 × 480	0.28 (36%)	0.22 (27%)	0.21 (26%)	0.21 (26%)
640 × 480	0.15 (18%)	0.13 (16%)	0.13 (16%)	0.13 (16%)
704 × 480	0.14 (18%)	0.12 (15%)	0.12 (15%)	0.12 (15%)
720 × 400	0.26 (33%)	0.15 (19%)	0.15 (18%)	0.15 (18%)
720 × 480	0.26 (33%)	0.15 (19%)	0.15 (18%)	0.15 (18%)
800 × 600	0.23 (28%)	0.15 (18%)	0.15 (19%)	0.15 (19%)
832 × 624	0.11 (14%)	0.11 (14%)	0.11 (14%)	0.11 (14%)
1024 × 768	0.09 (11%)	0.17 (21%)	0.17 (21%)	0.17 (21%)
1152 × 864	0.16 (20%)	0.09 (11%)	0.08 (10%)	0.08 (10%)
1280 × 720	0.08 (10%)	0.08 (10%)	0.09 (11%)	0.09 (11%)
1280 × 1024	0.08 (10%)	0.08 (10%)	0.08 (10%)	0.08 (10%)
1600 × 1200	0.16 (20%)	0.08 (10%)	0.09 (11%)	0.08 (10%)
1800 × 1440	0.18 (23%)	0.10 (13%)	0.10 (13%)	0.10 (13%)
1920 × 1080	0.08 (10%)	0.10 (13%)	0.09 (11%)	0.09 (12%)
1920 × 1200	0.08 (10%)	0.10 (13%)	0.09 (11%)	0.09 (12%)
Median	0.18 (22%)	0.15 (19%)	0.15 (18%)	0.15 (18%)
Mean	0.19 (24%)	0.16 (20%)	0.16 (20%)	0.16 (20%)
Standard deviation	0.09 (11%)	0.07 (9%)	0.07 (9%)	0.07 (9%)

Figure 6.9: Load and store bandwidth for vertical image access pattern. Layout is fixed at RSBCW and XOR levels are varied. Light and dark shadings in the **Image Size** column are used to indicate image sizes where the addition of XOR levels meaningfully (±5%) increased or decreased performance for *some* level of XORing.

XOR Levels	**0**	**1**	**2**	**3**
Peak Bandwidth	**0.8 GB/s**	**0.8 GB/s**	**0.8 GB/s**	**0.8 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.71 (89%)	0.74 (93%)	0.64 (80%)	0.64 (80%)
176 × 144	0.39 (49%)	0.53 (66%)	0.57 (71%)	0.57 (71%)
352 × 240	0.80 (100%)	0.39 (48%)	0.37 (46%)	0.38 (47%)
352 × 288	0.80 (100%)	0.39 (49%)	0.38 (47%)	0.38 (47%)
352 × 480	0.80 (100%)	0.39 (48%)	0.37 (46%)	0.38 (47%)
480 × 480	0.80 (100%)	0.47 (58%)	0.35 (43%)	0.36 (45%)
512 × 384	0.20 (25%)	0.40 (50%)	0.33 (41%)	0.33 (41%)
544 × 480	0.80 (100%)	0.51 (64%)	0.33 (41%)	0.34 (42%)
640 × 480	0.28 (35%)	0.26 (32%)	0.26 (32%)	0.26 (32%)
704 × 480	0.40 (50%)	0.24 (30%)	0.24 (30%)	0.25 (31%)
720 × 400	0.29 (36%)	0.35 (44%)	0.28 (35%)	0.28 (35%)
720 × 480	0.28 (36%)	0.36 (44%)	0.28 (35%)	0.28 (35%)
800 × 600	0.79 (99%)	0.24 (30%)	0.36 (45%)	0.34 (43%)
832 × 624	0.40 (50%)	0.27 (34%)	0.23 (28%)	0.23 (28%)
1024 × 768	0.20 (25%)	0.20 (25%)	0.37 (46%)	0.35 (44%)
1152 × 864	0.20 (25%)	0.18 (23%)	0.22 (27%)	0.22 (27%)
1280 × 720	0.14 (18%)	0.28 (35%)	0.20 (24%)	0.20 (25%)
1280 × 1024	0.14 (18%)	0.28 (35%)	0.20 (24%)	0.20 (25%)
1600 × 1200	0.40 (50%)	0.15 (18%)	0.23 (29%)	0.22 (28%)
1800 × 1440	0.17 (21%)	0.18 (22%)	0.21 (26%)	0.21 (26%)
1920 × 1080	0.20 (25%)	0.35 (43%)	0.20 (25%)	0.20 (24%)
1920 × 1200	0.20 (25%)	0.35 (43%)	0.20 (25%)	0.20 (24%)
Median	0.34 (42%)	0.35 (43%)	0.28 (35%)	0.28 (35%)
Mean	0.43 (53%)	0.34 (43%)	0.31 (39%)	0.31 (39%)
Standard deviation	0.26 (33%)	0.14 (17%)	0.12 (15%)	0.12 (15%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.34 (43%)	0.35 (44%)	0.33 (41%)	0.33 (41%)
176 × 144	0.27 (34%)	0.31 (39%)	0.32 (40%)	0.32 (40%)
352 × 240	0.36 (44%)	0.20 (25%)	0.20 (25%)	0.21 (26%)
352 × 288	0.36 (44%)	0.21 (26%)	0.21 (26%)	0.21 (26%)
352 × 480	0.36 (44%)	0.20 (26%)	0.20 (25%)	0.21 (26%)
480 × 480	0.35 (44%)	0.27 (33%)	0.18 (22%)	0.18 (23%)
512 × 384	0.10 (12%)	0.18 (22%)	0.16 (20%)	0.16 (20%)
544 × 480	0.36 (44%)	0.28 (35%)	0.16 (20%)	0.17 (21%)
640 × 480	0.13 (17%)	0.13 (16%)	0.12 (16%)	0.12 (16%)
704 × 480	0.18 (22%)	0.12 (15%)	0.12 (15%)	0.12 (15%)
720 × 400	0.13 (17%)	0.19 (23%)	0.13 (16%)	0.13 (16%)
720 × 480	0.13 (16%)	0.19 (23%)	0.13 (16%)	0.13 (16%)
800 × 600	0.35 (44%)	0.11 (14%)	0.18 (23%)	0.17 (21%)
832 × 624	0.18 (22%)	0.13 (16%)	0.10 (13%)	0.11 (13%)
1024 × 768	0.07 (8%)	0.09 (11%)	0.17 (21%)	0.17 (21%)
1152 × 864	0.09 (11%)	0.10 (12%)	0.09 (12%)	0.09 (12%)
1280 × 720	0.07 (8%)	0.13 (16%)	0.09 (12%)	0.10 (12%)
1280 × 1024	0.07 (8%)	0.13 (16%)	0.09 (12%)	0.10 (12%)
1600 × 1200	0.18 (22%)	0.07 (8%)	0.11 (14%)	0.11 (14%)
1800 × 1440	0.08 (10%)	0.09 (12%)	0.10 (12%)	0.10 (12%)
1920 × 1080	0.09 (11%)	0.17 (21%)	0.10 (12%)	0.09 (11%)
1920 × 1200	0.09 (11%)	0.17 (21%)	0.10 (12%)	0.09 (11%)
Median	0.16 (19%)	0.17 (21%)	0.13 (16%)	0.13 (16%)
Mean	0.20 (25%)	0.17 (22%)	0.15 (19%)	0.15 (19%)
Standard deviation	0.12 (15%)	0.08 (9%)	0.07 (8%)	0.07 (8%)

Figure 6.10: Load and store bandwidth for vertical image access pattern. Layout is fixed at RCSBW and XOR levels are varied. Light and dark shadings in the **Image Size** column are used to indicate image sizes where the addition of XOR levels meaningfully (±5%) increased or decreased performance for *some* level of XOR-ing. A mixed shading indicates that both some XOR level(s) increased performance and some XOR level(s) decreased performance. Light and dark shadings of the **Bandwidth** values are used to indicate data points in which the alternative layout of RCSBW meaningfully (±5%) increased or decreased performance relative to the corresponding data point for the default layout of RSBCW shown in Figure 6.9.

layout helps marginally for loads, and the degree of benefit diminishes with higher XOR levels. The benefits are further diminished for stores, and at higher XOR levels the average is slightly lower for the alternative layout for stores.

Next, we investigate how much the performance of a vertical access pattern suffers from each bank in VIRAM-1 consisting of only a single sub-bank. Figure 6.11 shows the performance of loads and stores as the number of sub-banks per bank is increased from 1 to 16. The benefits of sub-banks vary from case to case. Sometimes the memory conflicts are mostly sub-bank conflicts, but other times they are mostly bank conflicts. Increasing the number of sub-banks reduces sub-bank conflicts but does nothing for bank conflicts. Therefore, there are some cases in which doubling the number of sub-banks gives little or no increase in performance.

For example, the bandwidth for vertically accessing the image size of 128 × 96 does not improve at all with the addition of sub-banks, and the bandwidth for accessing the 176 × 144 image improves much less than all of the remaining cases. (The results for these images sizes are marked on Figure 6.11 in *italics*.) The stride is given by the image width. For the default RSBCW layout, a stride of 512 bytes advances to the next bank in the same wing. Strides less than 512 bytes potentially have bank conflicts. The situation is especially problematic for a stride of 128. A stride that is divisible by 32, but not by 64, alternates between the wings. But a stride that is divisible by 64, like 128, always goes to the same wing. Given that 4 addresses can be generated per cycle, Figure 6.12 (a) shows the access pattern that we would like to achieve, using the notation from Figure 2.1 to refer to the bank and column number. However, only one address can be issued per bank per cycle, and Figure 6.12 (b) shows the resulting access pattern. Instead of 4 addresses per cycle, we get an alternating pattern of 1, 1, and 2 address per cycle, which averages to 1.33. This is 33% of the peak of 4, which corresponds to the 33% of peak performance seen for both loads and stores for the image size of 128 × 96 in Figure 6.11.

At strides bigger than 128, but less than 512, there is the potential for some bank conflicts, although the actual results are better than for a stride of 128 because (1) less than 4 addresses are accessed before advancing to the next bank (4 × stride > 512 *iff* stride > 128); and (2) none of these image widths (176, 352, and 480) are divisible by 64, so they are able to take advantage of the banks in both wings, unlike the image width of 128. In fact, if the accesses alternate between both wings, then the threshold for which bank conflicts dominate is lowered from 512 bytes to 256 bytes. As a result, 176 × 144 is the only other small image size besides 128 × 96 for which the addition of sub-banks does not substantially increase bandwidth.

To summarize, the performance of smaller strides is not helped by increasing the number of sub-banks because a sufficient number of elements can be accessed from the same row spread across all of the banks before advancing to the next row. Sub-banks only potentially help when successive accesses to the same bank, but in a different row, are spaced closer together than the sub-bank busy time.

Another potential bank conflict problem exists for larger strides. Recall that large power of 2 length strides have the problem that every access is to the same bank. For instance, at a stride of 4096, if the first access was to bank 0, then every access would be to bank 0, producing a large number of bank conflicts. Since the default VIRAM configuration produces 4 addresses per cycle, however, there is the potential for similar conflicts for any stride where 4 consecutive elements span at least 4096 elements, i.e. strides greater than 1024. For instance, a stride of 2048 would alternate between banks 0, 4, 0, 4, 0, 4, . . . (using the notation of Figures 2.1 and 2.2), and half of the addresses would experience bank conflicts.

A stride of 512 advances to the next bank within the same wing. As described above, strides greater than 2 multiples of 512 are in danger of generating bank conflicts for a group of 4 addresses. Figure 6.13 (a) shows the repetitive pattern found at the beginning of an image with a stride of 1280 (as found in the image sizes 1280 × 720 and 1280 × 1024), which is exactly $2\frac{1}{2}$ multiples of 512. Figure 6.13 (b) shows the repetitive pattern found at the beginning of an image with a stride of 1920 (as found in the image sizes 1920 × 1080

Sub-banks **Peak Bandwidth**	**1** **0.8 GB/s**	**2** **0.8 GB/s**	**4** **0.8 GB/s**	**8** **0.8 GB/s**	**16** **0.8 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**				
128 × 96	*0.26 (33%)*	*0.27 (33%)*	*0.27 (33%)*	*0.27 (33%)*	*0.27 (33%)*
176 × 144	*0.41 (51%)*	*0.41 (51%)*	*0.42 (52%)*	*0.42 (52%)*	*0.42 (52%)*
352 × 240	0.60 (75%)	0.77 (96%)	0.77 (96%)	0.80 (100%)	0.80 (100%)
352 × 288	0.60 (75%)	0.78 (97%)	0.78 (97%)	0.78 (97%)	0.80 (100%)
352 × 480	0.60 (75%)	0.79 (99%)	0.79 (99%)	0.79 (99%)	0.80 (100%)
480 × 480	0.53 (66%)	0.78 (98%)	0.78 (98%)	0.78 (98%)	0.80 (100%)
512 × 384	0.40 (50%)	0.80 (100%)	0.80 (100%)	0.80 (100%)	0.80 (100%)
544 × 480	0.53 (66%)	0.80 (100%)	0.80 (100%)	0.80 (100%)	0.80 (100%)
640 × 480	0.32 (40%)	0.61 (77%)	0.80 (100%)	0.80 (100%)	0.80 (100%)
704 × 480	0.31 (38%)	0.57 (72%)	0.79 (99%)	0.79 (99%)	0.79 (99%)
720 × 400	0.55 (69%)	0.56 (70%)	0.79 (99%)	0.80 (100%)	0.80 (100%)
720 × 480	0.55 (69%)	0.56 (70%)	0.80 (100%)	0.80 (100%)	0.80 (100%)
800 × 600	0.48 (60%)	0.48 (60%)	0.79 (98%)	0.80 (100%)	0.80 (100%)
832 × 624	0.25 (31%)	0.48 (60%)	0.80 (99%)	0.80 (100%)	0.80 (100%)
1024 × 768	0.20 (25%)	0.40 (50%)	0.80 (100%)	0.80 (100%)	0.80 (100%)
1152 × 864	0.34 (42%)	0.34 (42%)	0.67 (83%)	0.80 (100%)	0.80 (100%)
1280 × 720	*0.18 (22%)*	*0.47 (59%)*	*0.47 (59%)*	*0.51 (64%)*	*0.51 (64%)*
1280 × 1024	*0.18 (22%)*	*0.48 (60%)*	*0.48 (60%)*	*0.52 (64%)*	*0.52 (64%)*
1600 × 1200	0.32 (40%)	0.32 (40%)	0.55 (69%)	0.80 (100%)	0.80 (100%)
1800 × 1440	0.39 (48%)	0.47 (59%)	0.47 (59%)	0.80 (100%)	0.80 (100%)
1920 × 1080	*0.17 (21%)*	*0.44 (55%)*	*0.45 (56%)*	*0.45 (56%)*	*0.45 (57%)*
1920 × 1200	*0.17 (21%)*	*0.44 (55%)*	*0.45 (57%)*	*0.45 (57%)*	*0.45 (57%)*
Median	0.36 (45%)	0.48 (60%)	0.78 (98%)	0.80 (100%)	0.80 (100%)
Mean	0.38 (47%)	0.55 (68%)	0.66 (82%)	0.70 (87%)	0.70 (88%)
Standard deviation	0.16 (19%)	0.17 (21%)	0.17 (22%)	0.17 (21%)	0.17 (21%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**				
128 × 96	*0.24 (30%)*	*0.27 (33%)*	*0.27 (33%)*	*0.27 (33%)*	*0.27 (33%)*
176 × 144	*0.36 (46%)*	*0.38 (47%)*	*0.42 (52%)*	*0.42 (52%)*	*0.42 (52%)*
352 × 240	0.27 (34%)	0.48 (61%)	0.71 (89%)	0.80 (100%)	0.80 (100%)
352 × 288	0.27 (34%)	0.49 (61%)	0.73 (91%)	0.73 (91%)	0.80 (100%)
352 × 480	0.27 (34%)	0.50 (63%)	0.76 (95%)	0.76 (95%)	0.80 (100%)
480 × 480	0.28 (36%)	0.48 (60%)	0.69 (86%)	0.75 (94%)	0.80 (100%)
512 × 384	0.18 (22%)	0.36 (44%)	0.71 (89%)	0.80 (100%)	0.80 (100%)
544 × 480	0.28 (36%)	0.63 (78%)	0.64 (80%)	0.80 (100%)	0.80 (100%)
640 × 480	0.15 (18%)	0.29 (36%)	0.55 (68%)	0.77 (96%)	0.80 (100%)
704 × 480	0.14 (18%)	0.26 (33%)	0.47 (59%)	0.76 (95%)	0.76 (95%)
720 × 400	0.26 (33%)	0.27 (33%)	0.51 (63%)	0.80 (100%)	0.80 (100%)
720 × 480	0.26 (33%)	0.27 (33%)	0.52 (65%)	0.78 (98%)	0.78 (98%)
800 × 600	0.23 (28%)	0.23 (28%)	0.47 (59%)	0.79 (98%)	0.79 (98%)
832 × 624	0.11 (14%)	0.23 (28%)	0.45 (57%)	0.80 (99%)	0.80 (100%)
1024 × 768	0.09 (11%)	0.18 (22%)	0.36 (44%)	0.71 (89%)	0.80 (100%)
1152 × 864	0.16 (20%)	0.16 (20%)	0.33 (41%)	0.78 (97%)	0.78 (97%)
1280 × 720	*0.08 (10%)*	*0.29 (36%)*	*0.29 (36%)*	*0.50 (63%)*	*0.50 (63%)*
1280 × 1024	*0.08 (10%)*	*0.29 (37%)*	*0.29 (37%)*	*0.52 (64%)*	*0.52 (64%)*
1600 × 1200	0.16 (20%)	0.16 (20%)	0.29 (37%)	0.79 (98%)	0.79 (98%)
1800 × 1440	0.18 (23%)	0.22 (28%)	0.22 (28%)	0.51 (64%)	0.80 (100%)
1920 × 1080	*0.08 (10%)*	*0.30 (38%)*	*0.36 (45%)*	*0.36 (45%)*	*0.45 (56%)*
1920 × 1200	*0.08 (10%)*	*0.30 (38%)*	*0.36 (46%)*	*0.37 (46%)*	*0.45 (57%)*
Median	0.18 (22%)	0.29 (36%)	0.46 (58%)	0.76 (95%)	0.79 (99%)
Mean	0.19 (24%)	0.32 (40%)	0.47 (59%)	0.66 (83%)	0.70 (87%)
Standard deviation	0.09 (11%)	0.12 (16%)	0.17 (21%)	0.18 (22%)	0.17 (21%)

Figure 6.11: Load and store bandwidth for vertical image access pattern. Layout is fixed at RSBCW, XORing is fixed at 0 levels, and sub-banks are varied. *Italics* is used to highlight cases of special note that are described further in the text.

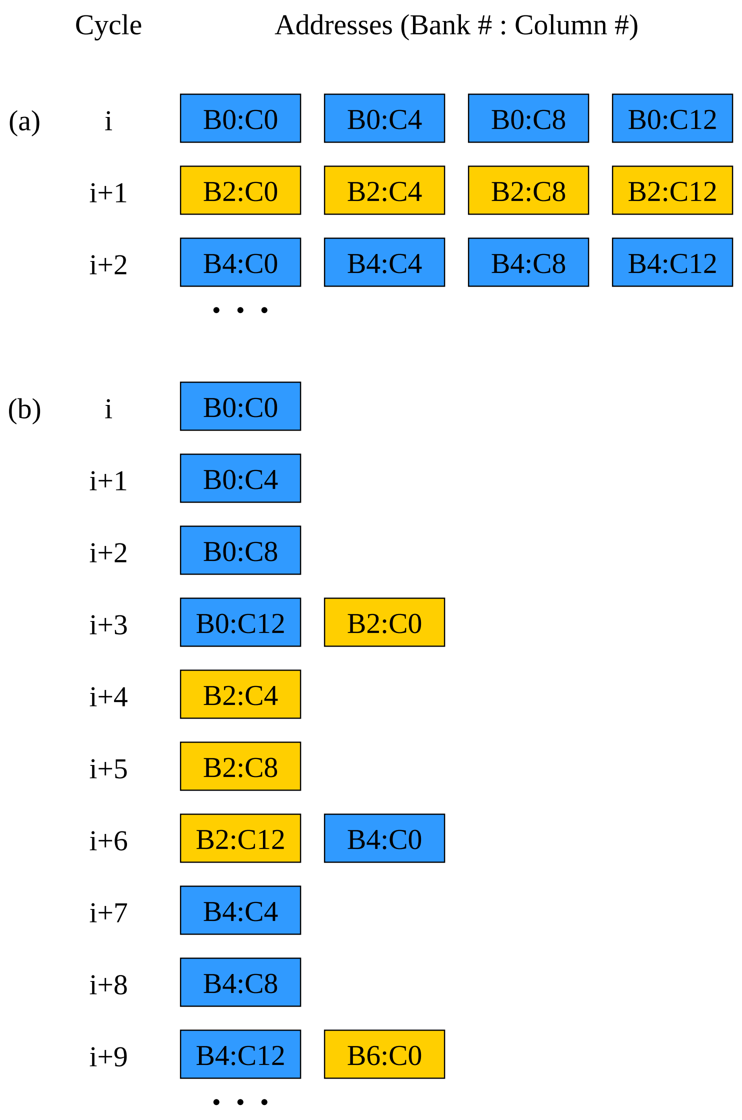


Figure 6.12: Effect of bank conflicts on 128 $\times$ 96 vertical image access pattern. Time advances downward, separately for each case. Each horizontal row contains one or more addresses issued during the same cycle. The numbering of banks and columns is consistent with that shown in Figures 2.1 and 2.2. The address mappings are for the default layout of RSBCW. Alternative shadings are used to indicate access to different banks. (a) shows the timing of addresses if there were no bank conflicts, processing 4 per cycle. (b) shows the resulting timing after resolving bank conflicts, as only 1 address can be processed per bank per cycle. Even with a sufficient number of sub-banks so that there are no sub-bank conflicts, this is the best possible timing given bank conflicts, resulting in 33% of peak performance.

and 1920 × 1200), which is exactly $3\frac{3}{4}$ multiples of 512. Both of these timings assume that there are no bank conflicts. Figures 6.13 (c) and (d) show the timings after resolving the bank conflicts.

Image Width (Stride) 1280

	Cycle	Addresses (Bank # : Column #)			
(a)	i	🟦 B0:C10	B6:C2	B10:C10	🟦 B0:C2
	i+1	🟨 B4:C10	B10:C2	B14:C10	🟨 B4:C2
	i+2	🟦 B8:C10	B14:C2	B2:C10	🟦 B8:C2
	i+3	🟨 B12:C10	B2:C2	B6:C10	🟨 B12:C2
	i+4	🟦 B0:C10	B6:C2	B10:C10	🟦 B0:C2
	i+5	🟨 B4:C10	B10:C2	B14:C10	🟨 B4:C2
	i+6	🟦 B8:C10	B14:C2	B2:C10	🟦 B8:C2
	i+7	🟨 B12:C10	B2:C2	B6:C10	🟨 B12:C2
	i+8	🟦 B0:C10	B6:C2	B10:C10	🟦 B0:C2
		. . .			

	Cycle	Addresses (Bank # : Column #)		
(c)	i	🟦 B0:C10	B6:C2	B10:C10
	i+1	🟦 B0:C2		
	i+2	🟨 B4:C10	B10:C2	B14:C10
	i+3	🟨 B4:C2		
	i+4	🟦 B8:C10	B14:C2	B2:C10
	i+5	🟦 B8:C2		
	i+6	🟨 B12:C10	B2:C2	B6:C10
	i+7	🟨 B12:C2		
	i+8	🟦 B0:C10	B6:C2	B10:C10
	i+9	🟦 B0:C2		
	i+10	🟨 B4:C10	B10:C2	B14:C10
	i+11	🟨 B4:C2		
	i+12	🟦 B8:C10	B14:C2	B2:C10
	i+13	🟦 B8:C2		
	i+14	🟨 B12:C10	B2:C2	B6:C10
	i+15	🟨 B12:C2		
	i+16	🟦 B0:C10	B6:C2	B10:C10
	i+17	🟦 B0:C2		
		. . .		

	Cycle	Addresses (Bank # : Column #)			
(e)	i	B6:C2	B10:C10	B0:C2	B4:C10
	i+1	B10:C2	B14:C10	B4:C2	B8:C10
	i+3	B14:C2	B2:C10	B8:C2	B12:C10
	i+4	B2:C2	B6:C10	B12:C2	B0:C10
	i+5	B6:C2	B10:C10	B0:C2	B4:C10
	i+6	B10:C2	B14:C10	B4:C2	B8:C10
	i+7	B14:C2	B2:C10	B8:C2	B12:C10
	i+8	B2:C2	B6:C10	B12:C2	B0:C10
	i+9	B6:C2	B10:C10	B0:C2	B4:C10
		. . .			

Image Width (Stride) 1920

	Cycle	Addresses (Bank # : Column #)			
(b)	i	🟦 B0:C10	B8:C6	🟦 B0:C2	B6:C14
	i+1	🟨 B14:C10	B6:C6	🟨 B14:C2	B4:C14
	i+2	🟦 B12:C10	B4:C6	🟦 B12:C2	B2:C14
	i+3	🟨 B10:C10	B2:C6	🟨 B10:C2	B0:C14
	i+4	🟦 B8:C10	B0:C6	🟦 B8:C2	B14:C14
	i+5	🟨 B6:C10	B14:C6	🟨 B6:C2	B12:C14
	i+6	🟦 B4:C10	B12:C6	🟦 B4:C2	B10:C14
	i+7	🟨 B2:C10	B10:C6	🟨 B2:C2	B8:C14
	i+8	🟦 B0:C10	B8:C6	🟦 B0:C2	B6:C14
		. . .			

	Cycle	Addresses (Bank # : Column #)	
(d)	i	🟦 B0:C10	B8:C6
	i+1	🟦 B0:C2	B6:C14
	i+2	🟨 B14:C10	B6:C6
	i+3	🟨 B14:C2	B4:C14
	i+4	🟦 B12:C10	B4:C6
	i+5	🟦 B12:C2	B2:C14
	i+6	🟨 B10:C10	B2:C6
	i+7	🟨 B10:C2	B0:C14
	i+8	🟦 B8:C10	B0:C6
	i+9	🟦 B8:C2	B14:C14
	i+10	🟨 B6:C10	B14:C6
	i+11	🟨 B6:C2	B12:C14
	i+12	🟦 B4:C10	B12:C6
	i+13	🟦 B4:C2	B10:C14
	i+14	🟨 B2:C10	B10:C6
	i+15	🟨 B2:C2	B8:C14
	i+16	🟦 B0:C10	B8:C6
	i+17	🟦 B0:C2	B6:C14
		. . .	

	Cycle	Addresses (Bank # : Column #)			
(f)	i	B8:C6	B0:C2	B6:C14	B14:C10
	i+1	B6:C6	B14:C2	B4:C14	B12:C10
	i+3	B4:C6	B12:C2	B2:C14	B10:C10
	i+4	B2:C6	B10:C2	B0:C14	B8:C10
	i+5	B0:C6	B8:C2	B14:C14	B6:C10
	i+6	B14:C6	B6:C2	B12:C14	B4:C10
	i+7	B12:C6	B4:C2	B10:C14	B2:C10
	i+8	B10:C6	B2:C2	B8:C14	B0:C10
	i+9	B8:C6	B0:C2	B6:C14	B14:C10
		. . .			

Figure 6.13: Effect of bank conflicts on 1280 × 720, 1280 × 1024, 1920 × 1080, and 1920 × 1200 vertical image access patterns. Time advances downward, separately for each case. Each horizontal row contains one or more addresses issued during the same cycle. The numbering of banks and columns is consistent with that shown in Figures 2.1 and 2.2. The address mappings are for the default layout of RSBCW. (a) and (b) show the timing of addresses if there were no bank conflicts, processing 4 per cycle, for the beginning of the 1280 wide and 1920 wide images, respectively. Shaded addresses show bank conflicts; alternating shadings are used to separate element groups. (c) and (d) show the resulting timing after resolving bank conflicts, as only 1 address can be processed per bank per cycle. The shaded addresses from (a) and (b) are left shaded to show how the conflicts were resolved. (e) and (f) show the timings that would result for this region of accesses if the image location was altered in memory by padding with an extra row. Assuming a sufficient number of sub-banks so that there are no sub-bank conflicts, the performance is 50% of peak for the accesses shown here in (c) and (d) and 100% of peak for the accesses shown in (e) and (f). Actual performance is somewhere within this range, and does not improve by padding the image, since this pattern is not representative of the entire image. See the text for complete details.

If this pattern repeated throughout the entire image, one could solve the problem simply by slightly changing the placement of the image in memory. Figures 6.13 (e) and (f) show the effect of shifting the first access in these patterns to the previous location of second access, corresponding to padding the image with an extra row at the beginning. This is as if the first address block on the first access is removed, and all other address blocks are rearranged so that there are again 4 addresses per cycle. In this case, assuming a sufficient number of sub-banks so that there are no sub-bank conflicts, there would be no bank conflicts and the performance would be expected to be 100% of peak, compared to the previous cases (c) and (d) in which the performance would be expected to be 50% of peak.

The actual performance achieved over the entire image, however, does not reflect either the 50% shown in (c) and (d) or the 100% shown in (e) and (f). Without altering the placement of the array in memory, the performance is approximately 65% for the 1280 × 720 and 1280 × 1024 image sizes and approximately 55% for the 1920 × 1080 and 1920 × 1200 image sizes, for a sufficient number of sub-banks, as marked in *italics* on Figure 6.11. Padding the image with an extra row at the beginning (not shown in any figure) does not noticeably change the performance. The reason for this is that the patterns shown in Figure 6.13 (c) and (d) only reflect the behavior for one column on the vertical image access pattern. Within this region, the performance is 50%. However, as the vertical image access advances from column to column, the precise repetitive pattern changes slightly, and there are several alternating regions whose performance is either 50% or 100% of peak, giving an average somewhere between those two values. Placing the image at a different location only moves the boundaries between those 50% and 100% performance regions; it does not noticeably change how large those regions are, and as such does not noticeably alter the overall bandwidth for accessing the entire image. The performance of a single strided access stream (one column) could be improved using this method, but the performance of vertically accessing the entire image is not.

It should be noted, however, that a more complex memory pipeline would be able to solve this problem. Since there are obvious ways of grouping 4 consecutive addresses so that there are no bank conflicts, if the vector memory functional unit were to send addresses to the memory system in the proper groupings, there would be no bank conflicts. However, in order to simplify the pipeline of VIRAM-1, all of the addresses for an element group must have their conflicts resolved before the next element group can be processed. Although element groups can split due to conflicts, it is not possible to dynamically move addresses from one existing element group to another. See [8] and [14] for more details.

The 1152 × 864, 1600 × 1200, and 1800 × 1440 image sizes, which have strides that are of $2\frac{1}{4}$, $3\frac{1}{8}$, and $3\frac{33}{64}$ multiples of 512 respectively, do not result in repetitive patterns where the same bank appears twice within any group of 4 consecutive addresses. As such, they do not experience bank conflicts, and they scale well with an increased number of sub-banks.

On average, Figure 6.11 shows that increasing the number of sub-banks can significantly help performance. The largest increases are going from 1 to 2 and from 2 to 4 sub-banks. There is less of an increase going from 4 to 8 sub-banks (more of an increase for stores than for loads), and there is very little difference going from 8 to 16 sub-banks for either loads or stores. While the VIRAM-1 implementation only has a single sub-bank per DRAM bank, 2, 4, or 8 sub-banks per bank are all likely values given 2 MB DRAM banks, with 4 sub-banks per bank perhaps being the most typical. Compared to 1 sub-bank, the mean performance for loads increases 1.5× for 2 sub-banks, 1.8× for 4 sub-banks, 1.9× for 8 sub-banks, and 1.9× for 16 sub-banks. The performance for stores increases 1.7× for 2 sub-banks, 2.5× for 4 sub-banks, 3.4× for 8 sub-banks, and 3.6× for 16 sub-banks. Since most image sizes benefit significantly from sub-banks, but a few (those marked in *italics*) are dominated by bank conflicts and do not, this results in a wide variance, and with a sufficiently large number of sub-banks these low-lying outliers cause the median to be substantially greater than the mean.

The addition of address hashing via XORing of bank selection bits helped the performance of large power of 2 length strides by increasing the periodicity of the bank access pattern. Such an addition can also help reduce, although not entirely eliminate, the bank conflicts experienced by some large image sizes for the vertical image access pattern. Figure 6.14 shows the results for the same configurations as Figure 6.11, but with the addition of 1 XOR level. Adding XORing does improve the cases which were specifically targeted (and marked in *italics*); for 16 sub-banks, the mean improvement amongst the image sizes of 1280 × 720, 1280 × 1024, 1920 × 1080, and 1920 × 1200, is 1.20× for loads and 1.18× for stores. However, the mean improvement for these cases is somewhat less, only 1.12× for loads and 1.13× for stores, for the default configuration of 1 sub-bank, when sub-bank conflicts as well as bank conflicts are limiting performance. Additionally, although adding XORing can increase the performance of some cases, it also noticeably decreases the performance of others. Figure 6.9 showed this for 1 sub-bank for various levels of XORing; comparing Figure 6.14 to Figure 6.11 shows this for 1 XOR level for higher levels of sub-banks as well. The mean bandwidth for 1 XOR level compared to 0 XOR levels decreases by 0.79×, 0.87×, 0.95×, 0.94×, and 0.94× for 1, 2, 4, 8, and 16 sub-banks respectively for loads, and by 0.84×, 0.81×, 0.94×, 0.94×, and 0.94× for stores.

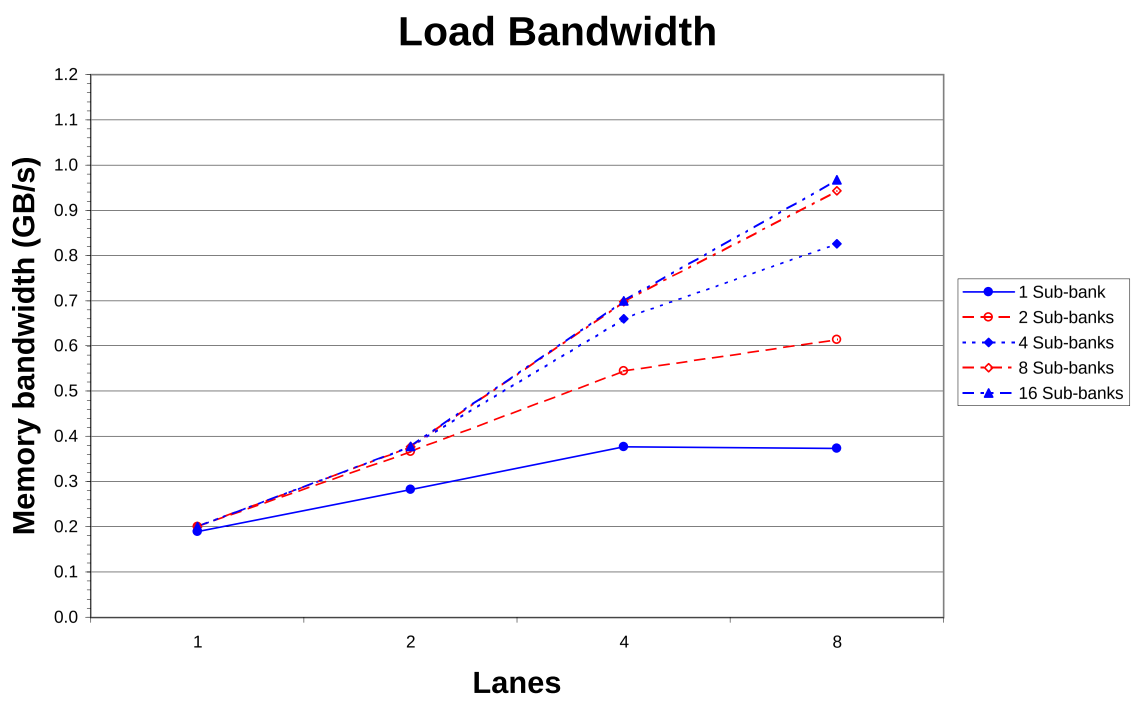
Next, how well the performance of a vertical access pattern scales as the number of lanes is scaled up and down was investigated. Figure 6.15 shows the mean bandwidth as the number of lanes is scaled from 1 to 8, for 1 to 16 sub-banks. The peak bandwidth varies from 0.2 GB/s to 1.6 GB/s as the number of lanes varies from 1 to 8. With only 1 sub-bank, the performance does not scale very well due to sub-bank conflicts. Compared to 1 lane, the performance for loads increases by 1.5×, 2.0×, and 1.9× for 2, 4, and 8 lanes respectively. For stores, the performance increases by 1.1×, 1.4×, and 1.4× for 2, 4, and 8 lanes respectively compared to 1 lane. As expected, the performance scales better when there are more sub-banks. Again, the largest difference in scaling comes from increasing from from 1 to 2 and from 2 to 4 sub-banks. For loads, there is less of an increase when going from 4 to 8 sub-banks, and very little difference for both loads and stores from 8 to 16 sub-banks. Even with a sufficient number of sub-banks, there is still not perfect scaling, since there are still bank conflicts. For 16 sub-banks, the performance increases by 1.9×, 3.5×, and 4.9× for loads and by 1.9×, 3.5×, and 4.8× for stores for 2, 4, and 8 lanes respectively compared to 1 lane.

As was mentioned earlier, VIRAM-1 limits the number of addresses that can be generated per memory unit to 4 per cycle, regardless of the `vpw`, due to the total resources required for generating and processing addresses. Figure 6.16 shows the effects of increasing the address generation resources up to the full 16 that could be possible given 4 lanes and `vpw` = 16, for 1 to 16 sub-banks. The peak bandwidth varies from 0.8 GB/s to 3.2 GB/s as the address generation resources vary from 4 to 16. With only 1 sub-bank, increasing the address generation resources does not help significantly, due to memory conflicts. The mean load bandwidth only increases by a factor of 1.1× and the mean store bandwidth by 1.1×, even as the number of address generators is increased by a factor of 4. As the number of sub-banks is increased, the benefit from increasing address generation resources increases. However, the benefits are still limited, due to bank conflicts, and the mean load bandwidth increases by a factor of 1.7× and the mean store bandwidth by 1.6×. With the full 16 addresses possible per cycle and 16 sub-banks, the mean measured bandwidths of 1.17 GB/s for loads and 1.12 GB/s for stores are indeed more than the 0.8 GB/s peak bandwidth with only 4 addresses per cycle, but these are still far short (36% for loads and 35% for stores) of the 3.2 GB/s peak bandwidth possible with 16 addresses per cycle.

Finally, by comparing any of the data for stores to the corresponding data for loads in Figures 6.9, 6.10, 6.11, 6.14, 6.15, and 6.16, it can be seen that in general load performance is better than store performance. This performance differential is because stores keep the sub-bank busy for a longer period of time than do loads, as explained previously in Section 2.2. As can be seen from Figures 6.15 and 6.16, however, as the number of sub-banks is increased, there are less sub-bank conflicts, the sub-bank busy time is less of an issue, and the differential between the load and store performances decreases. For all of these cases, the mean store performance averages only 55% of the load performance at 1 sub-bank, but 64% at 2 sub-banks, 73% at 4 sub-banks, 92% at 8 sub-banks, and 99% of the load performance at 16 sub-banks.

Sub-banks **Peak Bandwidth**	**1** **0.8 GB/s**	**2** **0.8 GB/s**	**4** **0.8 GB/s**	**8** **0.8 GB/s**	**16** **0.8 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**				
128 × 96	0.27 (33%)	0.27 (33%)	0.27 (33%)	0.27 (33%)	0.27 (33%)
176 × 144	0.40 (50%)	0.42 (52%)	0.42 (52%)	0.42 (52%)	0.42 (52%)
352 × 240	0.43 (54%)	0.71 (89%)	0.73 (92%)	0.77 (96%)	0.77 (96%)
352 × 288	0.43 (54%)	0.72 (90%)	0.75 (93%)	0.75 (93%)	0.77 (96%)
352 × 480	0.43 (54%)	0.73 (91%)	0.77 (96%)	0.77 (96%)	0.77 (96%)
480 × 480	0.47 (58%)	0.63 (79%)	0.78 (97%)	0.78 (97%)	0.79 (99%)
512 × 384	0.33 (41%)	0.62 (78%)	0.74 (93%)	0.74 (93%)	0.74 (93%)
544 × 480	0.43 (53%)	0.64 (80%)	0.79 (99%)	0.79 (99%)	0.79 (99%)
640 × 480	0.27 (34%)	0.51 (64%)	0.73 (91%)	0.73 (91%)	0.73 (91%)
704 × 480	0.25 (31%)	0.46 (58%)	0.71 (89%)	0.71 (89%)	0.71 (89%)
720 × 400	0.31 (39%)	0.54 (67%)	0.77 (96%)	0.77 (96%)	0.77 (96%)
720 × 480	0.31 (39%)	0.53 (67%)	0.76 (94%)	0.77 (96%)	0.77 (96%)
800 × 600	0.30 (38%)	0.53 (66%)	0.75 (94%)	0.76 (95%)	0.76 (95%)
832 × 624	0.23 (29%)	0.41 (51%)	0.69 (86%)	0.69 (86%)	0.69 (86%)
1024 × 768	0.37 (46%)	0.37 (46%)	0.74 (93%)	0.80 (100%)	0.80 (100%)
1152 × 864	0.18 (23%)	0.36 (45%)	0.58 (72%)	0.58 (72%)	0.58 (72%)
1280 × 720	*0.18 (22%)*	*0.32 (40%)*	*0.48 (61%)*	*0.57 (71%)*	*0.57 (71%)*
1280 × 1024	*0.18 (22%)*	*0.32 (40%)*	*0.49 (61%)*	*0.57 (72%)*	*0.57 (72%)*
1600 × 1200	0.18 (22%)	0.32 (39%)	0.43 (54%)	0.54 (68%)	0.54 (68%)
1800 × 1440	0.21 (27%)	0.34 (43%)	0.41 (51%)	0.58 (73%)	0.58 (73%)
1920 × 1080	*0.21 (26%)*	*0.36 (45%)*	*0.58 (72%)*	*0.58 (72%)*	*0.58 (72%)*
1920 × 1200	*0.21 (26%)*	*0.36 (46%)*	*0.58 (72%)*	*0.58 (72%)*	*0.58 (72%)*
Median	0.29 (36%)	0.44 (55%)	0.72 (90%)	0.72 (90%)	0.72 (90%)
Mean	0.30 (37%)	0.48 (60%)	0.63 (79%)	0.66 (82%)	0.66 (83%)
Standard deviation	0.10 (12%)	0.15 (18%)	0.15 (19%)	0.14 (17%)	0.14 (17%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**				
128 × 96	0.26 (33%)	0.27 (33%)	0.27 (33%)	0.27 (33%)	0.27 (33%)
176 × 144	0.33 (41%)	0.41 (51%)	0.42 (52%)	0.42 (52%)	0.42 (52%)
352 × 240	0.22 (28%)	0.41 (51%)	0.68 (85%)	0.77 (96%)	0.77 (96%)
352 × 288	0.22 (28%)	0.42 (52%)	0.70 (88%)	0.70 (88%)	0.77 (96%)
352 × 480	0.22 (28%)	0.43 (53%)	0.74 (92%)	0.74 (92%)	0.77 (96%)
480 × 480	0.23 (29%)	0.34 (42%)	0.62 (78%)	0.75 (94%)	0.79 (99%)
512 × 384	0.16 (20%)	0.32 (40%)	0.64 (80%)	0.74 (93%)	0.74 (93%)
544 × 480	0.22 (27%)	0.35 (44%)	0.58 (73%)	0.79 (99%)	0.79 (99%)
640 × 480	0.13 (16%)	0.25 (32%)	0.49 (61%)	0.70 (88%)	0.73 (91%)
704 × 480	0.12 (15%)	0.23 (29%)	0.46 (58%)	0.69 (86%)	0.69 (86%)
720 × 400	0.15 (19%)	0.27 (34%)	0.46 (58%)	0.77 (96%)	0.77 (96%)
720 × 480	0.15 (19%)	0.27 (33%)	0.46 (57%)	0.76 (94%)	0.76 (94%)
800 × 600	0.15 (18%)	0.26 (32%)	0.41 (51%)	0.75 (93%)	0.75 (93%)
832 × 624	0.11 (14%)	0.20 (25%)	0.39 (49%)	0.69 (86%)	0.69 (86%)
1024 × 768	0.17 (21%)	0.17 (21%)	0.34 (43%)	0.71 (89%)	0.80 (100%)
1152 × 864	0.09 (11%)	0.18 (22%)	0.36 (45%)	0.57 (71%)	0.57 (71%)
1280 × 720	*0.08 (10%)*	*0.17 (21%)*	*0.28 (35%)*	*0.56 (70%)*	*0.56 (70%)*
1280 × 1024	*0.08 (10%)*	*0.17 (21%)*	*0.28 (35%)*	*0.57 (72%)*	*0.57 (72%)*
1600 × 1200	0.08 (10%)	0.16 (20%)	0.22 (27%)	0.54 (67%)	0.54 (67%)
1800 × 1440	0.10 (13%)	0.16 (20%)	0.20 (24%)	0.45 (56%)	0.58 (73%)
1920 × 1080	*0.10 (13%)*	*0.20 (24%)*	*0.36 (45%)*	*0.36 (45%)*	*0.56 (71%)*
1920 × 1200	*0.10 (13%)*	*0.20 (24%)*	*0.36 (46%)*	*0.37 (46%)*	*0.57 (71%)*
Median	0.15 (19%)	0.26 (32%)	0.41 (51%)	0.69 (87%)	0.71 (89%)
Mean	0.16 (20%)	0.26 (33%)	0.44 (55%)	0.62 (78%)	0.66 (82%)
Standard deviation	0.07 (9%)	0.09 (11%)	0.16 (20%)	0.16 (20%)	0.14 (17%)

Figure 6.14: Load and store bandwidth for vertical image access pattern. Layout is fixed at RSBCW, XORing is fixed at 1 level, and sub-banks are varied. *Italics* is used to highlight cases of special note that are described further in the text. Light and dark shadings of the **Bandwidth** values are used to indicate data points in which the addition of 1 XOR level meaningfully (±5%) increased or decreased performance relative to the corresponding data point for the default of 0 XOR levels shown in Figure 6.11.



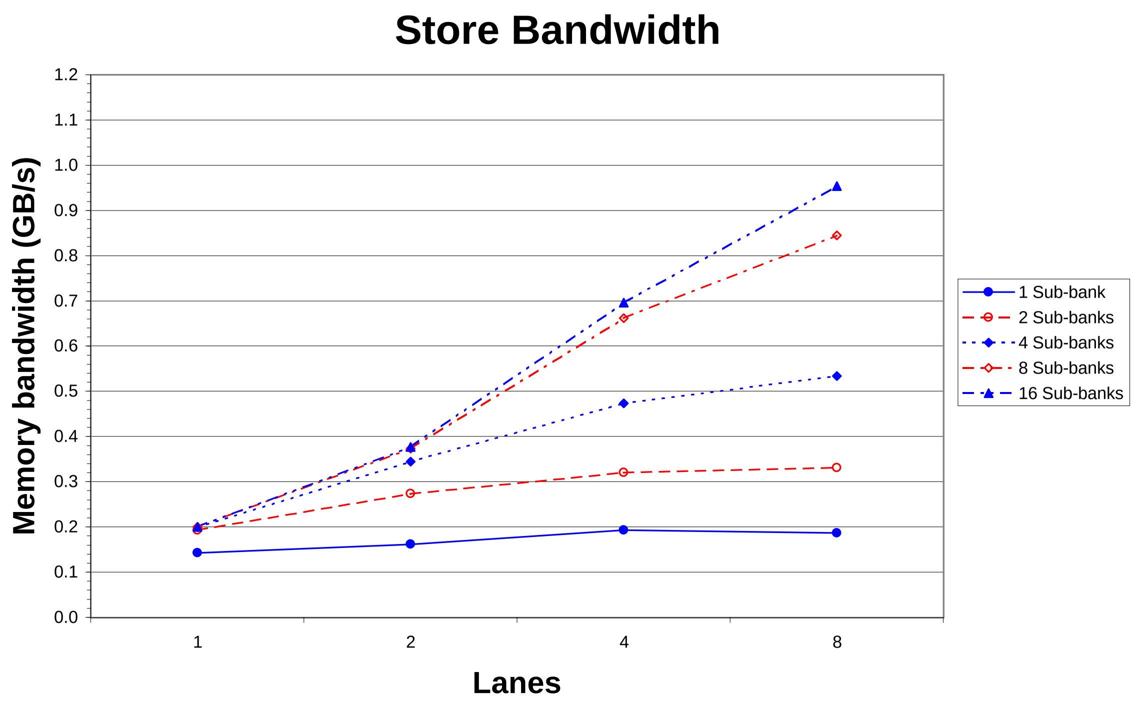
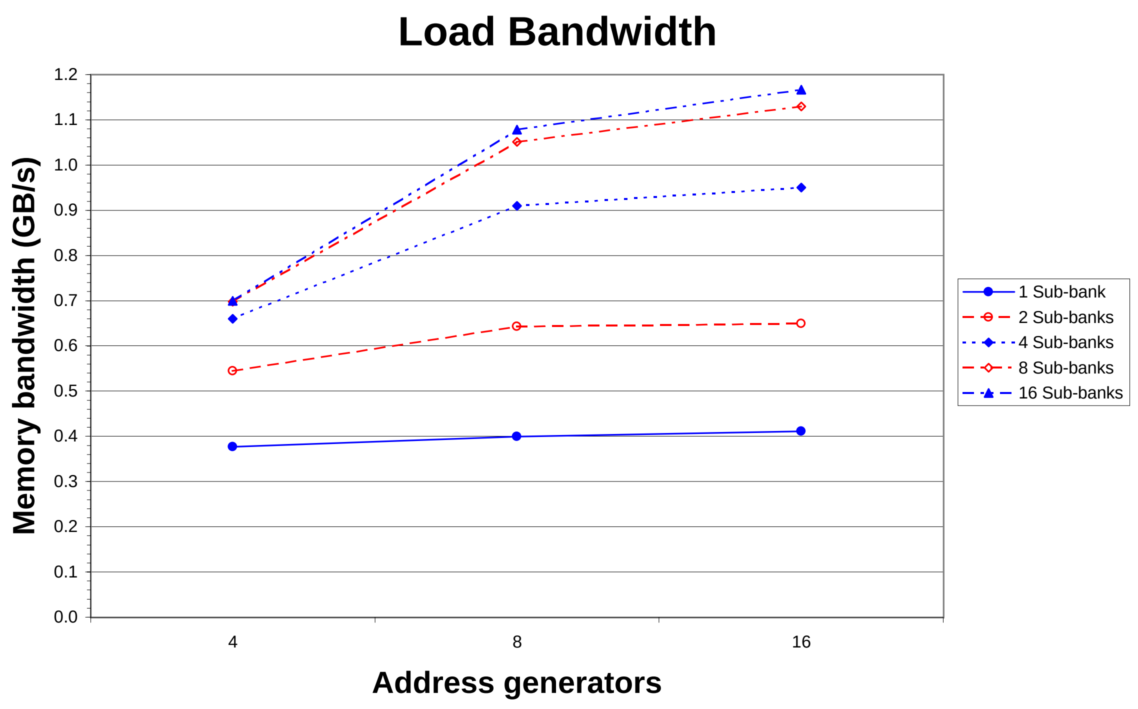


Figure 6.15: Mean load and store bandwidth for vertical image access pattern. Layout is fixed at RSBCW, XORing is fixed at 0 levels, and lanes and sub-banks are varied. Peak bandwidth is 0.2 GB/s, 0.4 GB/s, 0.8 GB/s and 1.6 GB/s for 1, 2, 4, and 8 lanes respectively. Figures A.2 through A.6 in the Appendix show the data from which these means are derived.



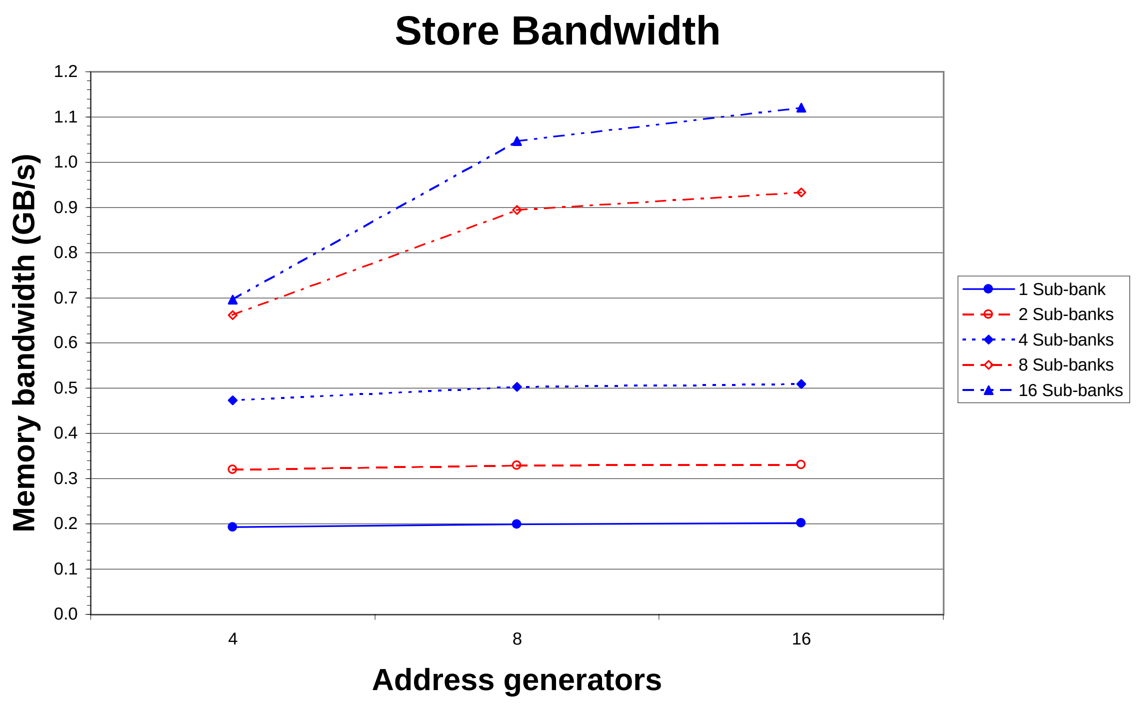


Figure 6.16: Mean load and store bandwidth for vertical image access pattern. Layout is fixed at RSBCW, XORing is fixed at 0 levels, and address generation resources and sub-banks are varied. Peak bandwidth is 0.8 GB/s, 1.6 GB/s, and 3.2 GB/s for 4, 8, and 16 address generators respectively. Figures A.7 through A.11 in the Appendix show the data from which these means are derived.

6.3 8 × 8 Blocked Image Access Pattern

Figure 6.17 shows the performance of loads and stores in a blocked image access pattern of 8 × 8 blocks. Although the blocked accesses involve a series of unit strides (8 unit strides each of vector length 8 per block), the performance is significantly diminished from that of the unit stride accesses in a horizontal access pattern. The short vectors used for the blocked access pattern are responsible for this performance degradation. At `vpw` = 16 bits and 4 lanes there are 16 VPs available. With a vector length of 8, however, only half of the VPs can be used, cutting the effective peak bandwidth in half. Because each instruction is completely covered by a single element group, however, there is not enough vector instruction bandwidth available to keep both memory units busy.[2] Only using one of the two available memory units effectively cuts the peak bandwidth in half again. Therefore, despite the fact that the peak bandwidth that the hardware provides for loading and storing 8-bit data with `vpw` = 16 bits is 6.4 GB/s, the most that we can hope to achieve for the blocked access pattern is 1.6 GB/s, or 25% of peak.

Unrolling the inner loop helps to use more of the vector issue bandwidth, but it is not possible to unroll the loop sufficiently to eliminate all of the empty slots due to insufficient vector issue bandwidth, and the peak is less than the 25% that we could hope for. With the original (not unrolled) loop, one iteration loads an entire 8 × 8 block, placing each row into one of 8 registers. The most that the loop can be unrolled is therefore 4 times, since this consumes all of the 32 vector registers.

Even with the loop unrolled, having the data aligned or not makes only a marginal difference. Contrast this with Figure 6.1 for the horizontal image access pattern in which data alignment makes the difference between 80% and 100% of peak performance. An unaligned unit stride access only needs extra cycles when the access crosses a boundary equal in size to the physical width spanning all of the lanes (256 bits for 4 lanes). Since each blocked unit stride is only 64 bits (8 elements of 8 bits each), even if the array is not 256-bit aligned, any individual access may not cross the boundary, and in the worst case would only matter for 1 of every 4 accesses. Also, just as was true for the horizontal access pattern without loop unrolling, keeping the memory unit busy for an extra cycle may not matter simply because there are already empty cycles due to insufficient issue bandwidth.

Keeping the data aligned and the loop unrolled 4 times, Figure 6.18 shows the load and store performance as the number of lanes is scaled from 1 to 8. The problems identified above of insufficient vector instruction bandwidth and not being able to use all of physical hardware depend on the vector length relative to what is supplied by the hardware. As the number of lanes is increased, the problems associated with short vectors become relatively more severe. As stated above, at 4 lanes we can only use half of the available VPs (8 of 16). At 8 lanes, the situation is worse, and only one fourth of the available VPs (8 of 32) can be used, since the vector length stays constant at 8 despite the doubling of the available hardware resources. The problem of there not being sufficient instruction issue bandwidth to use both memory units is true at 2, 4, and 8 lanes. When the hardware is scaled down to 1 physical lane the relative length of the vector (8) is long enough that both memory units can be used. The peak bandwidth provided by the hardware for 8-bit data at `vpw` = 16 bits is 1.6, 3.2, 6.4, and 12.8 GB/s for 1, 2, 4, and 8 lanes respectively. The effective peak bandwidth that we can hope to achieve, however, stays constant at 1.6 GB/s even as the number of lanes is scaled from 1 to 8, which is 100%, 50%, 25%, and 12.5% of the peak respectively.

On top of these difficulties, since the unit stride loads and stores involved in a blocked access pattern are not consecutive, there are similar conflicts to those encountered with non-unit stride access streams. Some of the cases experience sub-bank conflicts. For 2, 4, and 8 lanes, since there is not enough vector issue bandwidth to use more than one memory unit, there is only one address issued on every cycle, resulting

[2]Unlike the issue bandwidth problems noted earlier in Section 6.1, this is entirely a real problem and not a simulation artifact. The difference is that this is referring to the limitations of issuing instructions to the vector functional units, and not the issue width of the scalar core. Increasing this beyond one per cycle would imply a superscalar vector unit, which would significantly increase control complexity and is not planned for VIRAM-1.

Loop Unrolled	**1**	**1**	**2**	**2**	**4**	**4**
Data Aligned?	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**
Peak Bandwidth	**6.4 GB/s**	**6.4 GB/s**	**6.4 GB/s**	**6.4 GB/s**	**6.4 GB/s**	**6.4 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**					
128 × 96	1.1 (16%)	1.1 (16%)	1.2 (18%)	1.2 (18%)	1.3 (20%)	1.3 (20%)
176 × 144	1.1 (17%)	1.1 (17%)	1.2 (19%)	1.2 (19%)	1.3 (20%)	1.3 (20%)
352 × 240	1.1 (17%)	1.1 (17%)	1.3 (20%)	1.3 (20%)	1.4 (22%)	1.4 (22%)
352 × 288	1.1 (17%)	1.1 (17%)	1.3 (20%)	1.3 (20%)	1.4 (22%)	1.4 (22%)
352 × 480	1.1 (17%)	1.1 (17%)	1.3 (20%)	1.3 (20%)	1.4 (22%)	1.4 (22%)
480 × 480	1.1 (18%)	1.1 (18%)	1.3 (20%)	1.3 (20%)	1.4 (22%)	1.4 (22%)
512 × 384	1.1 (18%)	1.1 (18%)	1.3 (20%)	1.3 (20%)	1.4 (22%)	1.4 (22%)
544 × 480	1.1 (18%)	1.1 (18%)	1.3 (20%)	1.3 (20%)	1.4 (22%)	1.4 (22%)
640 × 480	1.1 (18%)	1.1 (18%)	1.3 (20%)	1.3 (20%)	1.3 (20%)	1.3 (20%)
704 × 480	1.1 (18%)	1.1 (18%)	1.3 (20%)	1.3 (20%)	1.3 (21%)	1.3 (21%)
720 × 400	1.1 (18%)	1.1 (18%)	1.3 (21%)	1.3 (21%)	1.4 (22%)	1.4 (22%)
720 × 480	1.1 (18%)	1.1 (18%)	1.3 (21%)	1.3 (21%)	1.4 (22%)	1.4 (22%)
800 × 600	1.1 (18%)	1.1 (18%)	1.3 (21%)	1.3 (21%)	1.4 (22%)	1.4 (22%)
832 × 624	1.1 (18%)	1.1 (18%)	1.3 (21%)	1.3 (21%)	1.4 (22%)	1.4 (22%)
1024 × 768	1.1 (18%)	1.1 (18%)	1.3 (21%)	1.3 (21%)	1.4 (22%)	1.4 (22%)
1152 × 864	1.1 (18%)	1.1 (18%)	1.3 (20%)	1.3 (20%)	1.3 (20%)	1.3 (20%)
1280 × 720	1.2 (18%)	1.2 (18%)	1.3 (21%)	1.3 (21%)	1.3 (21%)	1.3 (21%)
1280 × 1024	1.2 (18%)	1.2 (18%)	1.3 (21%)	1.3 (21%)	1.3 (21%)	1.3 (21%)
1600 × 1200	1.2 (18%)	1.2 (18%)	1.3 (21%)	1.3 (21%)	1.4 (23%)	1.4 (23%)
1800 × 1440	1.2 (18%)	1.2 (18%)	1.3 (21%)	1.3 (21%)	1.4 (22%)	1.4 (22%)
1920 × 1080	1.1 (17%)	1.1 (17%)	1.1 (17%)	1.1 (17%)	1.1 (17%)	1.1 (17%)
1920 × 1200	1.1 (17%)	1.1 (17%)	1.1 (17%)	1.1 (17%)	1.1 (17%)	1.1 (17%)
Median	1.1 (18%)	1.1 (18%)	1.3 (20%)	1.3 (20%)	1.4 (22%)	1.4 (22%)
Mean	1.1 (18%)	1.1 (18%)	1.3 (20%)	1.3 (20%)	1.4 (21%)	1.4 (21%)
Standard deviation	0.03 (0%)	0.03 (0%)	0.07 (1%)	0.07 (1%)	0.09 (1%)	0.09 (1%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**					
128 × 96	1.1 (16%)	1.1 (16%)	1.2 (18%)	1.2 (18%)	1.3 (20%)	1.3 (20%)
176 × 144	1.1 (17%)	1.1 (17%)	1.2 (19%)	1.2 (19%)	1.3 (20%)	1.3 (20%)
352 × 240	1.1 (17%)	1.1 (17%)	1.3 (20%)	1.3 (20%)	1.4 (22%)	1.4 (22%)
352 × 288	1.1 (17%)	1.1 (17%)	1.3 (20%)	1.3 (20%)	1.4 (22%)	1.4 (22%)
352 × 480	1.1 (17%)	1.1 (17%)	1.3 (20%)	1.3 (20%)	1.4 (22%)	1.4 (22%)
480 × 480	1.1 (18%)	1.1 (18%)	1.3 (20%)	1.3 (20%)	1.4 (22%)	1.4 (22%)
512 × 384	1.1 (18%)	1.1 (18%)	1.3 (20%)	1.3 (20%)	1.4 (22%)	1.4 (22%)
544 × 480	1.1 (18%)	1.1 (18%)	1.3 (20%)	1.3 (20%)	1.4 (21%)	1.4 (21%)
640 × 480	0.8 (12%)	0.8 (12%)	0.8 (12%)	0.8 (12%)	0.8 (12%)	0.8 (12%)
704 × 480	0.8 (12%)	0.8 (12%)	0.8 (12%)	0.8 (12%)	0.8 (12%)	0.8 (12%)
720 × 400	1.1 (18%)	1.1 (18%)	1.2 (19%)	1.2 (19%)	1.2 (19%)	1.2 (19%)
720 × 480	1.1 (18%)	1.1 (18%)	1.2 (19%)	1.2 (19%)	1.2 (19%)	1.2 (19%)
800 × 600	1.1 (18%)	1.1 (18%)	1.3 (21%)	1.3 (21%)	1.3 (21%)	1.3 (21%)
832 × 624	0.8 (12%)	0.8 (12%)	0.8 (12%)	0.8 (12%)	0.8 (12%)	0.8 (12%)
1024 × 768	0.8 (12%)	0.8 (12%)	0.8 (12%)	0.8 (12%)	0.8 (12%)	0.8 (12%)
1152 × 864	0.9 (14%)	0.9 (14%)	0.9 (14%)	0.9 (14%)	0.9 (14%)	0.9 (14%)
1280 × 720	0.7 (11%)	0.7 (11%)	0.7 (11%)	0.7 (11%)	0.7 (11%)	0.7 (11%)
1280 × 1024	0.7 (11%)	0.7 (11%)	0.7 (11%)	0.7 (11%)	0.7 (11%)	0.7 (11%)
1600 × 1200	0.8 (13%)	0.8 (13%)	0.8 (13%)	0.8 (13%)	0.8 (13%)	0.8 (13%)
1800 × 1440	1.1 (17%)	1.1 (17%)	1.2 (18%)	1.2 (18%)	1.2 (19%)	1.2 (19%)
1920 × 1080	0.6 (10%)	0.6 (10%)	0.6 (10%)	0.6 (10%)	0.6 (10%)	0.6 (10%)
1920 × 1200	0.6 (10%)	0.6 (10%)	0.6 (10%)	0.6 (10%)	0.6 (10%)	0.6 (10%)
Median	1.1 (17%)	1.1 (17%)	1.2 (18%)	1.2 (18%)	1.2 (19%)	1.2 (19%)
Mean	0.9 (15%)	0.9 (15%)	1.0 (16%)	1.0 (16%)	1.1 (17%)	1.1 (17%)
Standard deviation	0.20 (3%)	0.20 (3%)	0.27 (4%)	0.27 (4%)	0.31 (5%)	0.31 (5%)

Figure 6.17: Load and store bandwidth for blocked image access pattern. Alignment of data refers to 256-bit boundaries. Maximum possible bandwidth for this benchmark is 1.6 GB/s, 25% of the peak provided by the hardware.

in no bank conflicts. With 1 lane, however, where the relative vector length is longer and both memory units are utilized, there is a new problem. Only a single unit stride access can be sent to each wing on each cycle. As has been mentioned earlier, the default layout of RSBCW has been chosen so that two unit stride streams can peacefully coexist by alternating wings. That model is broken here, however, since each unit stride access is so short. Two consecutive accesses, which are separated by the image width if they are in the same 8 × 8 block, will often map to the same wing, causing conflicts. For most image widths, two consecutive accesses in same block do map to the same wing. The only image size for which they map to different wings is 1800 × 1440.

For the default layout of RSBCW, a distance between two accesses of 8 bytes maps to the opposing wing, and a distance of 16 bytes maps to the same wing. Consecutive accesses will therefore map to the same wing if the distance, given by the image width, is divisible by 16. They will map to different wings if the image width is only divisible by 8. It happens to be that the image size of 1800 × 1440 is the only one in which the image width is only divisible by 8; for all other image sizes studied, the image width is divisible by 16. The result of this is that the bandwidth for 1800 × 1440 for 1 lane is considerably higher than for all of the other image sizes. This effect is much more pronounced for loads than it is for stores. The outlying 1800 × 1440 case achieves a bandwidth of 1.3 GB/s (82%) for loads (marked in *italics*) compared to the mean for all images sizes of 0.9 GB/s (55%) for loads. Although the bank conflicts for stores are significantly reduced for the case of 1800 × 1440 for 1 lane, there are also a significant number of sub-bank conflicts. Therefore, the bandwidth of 0.9 GB/s (58%) for stores, which have a longer sub-bank busy time than loads, is not drastically higher then the mean for all the image sizes of 0.8 GB/s (50%).

The net effect of these various complications is that the actual bandwidth achieved increases somewhat from 1 to 2 lanes. At higher number of lanes, the mean bandwidth does not increase significantly, and in some cases actually decreases. Compared to 1 lane, the mean performance for loads is 1.6× higher for 2 lanes, 1.6× higher for 4 lanes, and 1.5× higher for 8 lanes. For stores, the mean performance for loads is 1.3× higher for 2 lanes, 1.3× higher for 4 lanes, and 1.3× higher for 8 lanes.

6.4 Randomized Image Access Pattern

A randomized image access pattern was used to study the behavior of indexed memory references. While often associated with sparse matrix calculations for scientific computing, indexed operations will occur in any application involving pointer jumping. One example of a vectorizable multimedia application using indexed operations is 3D rasterization.

Figure 6.19 shows the load and store performance for a series of randomized accesses distributed throughout an image array. In general the image size is not a significant factor in the measured bandwidth; each case consists of 10000 loads or stores, and the variance is small across the images sizes. One might have expected that larger images might have slightly higher performance, because the addresses are spread out more. The results obtained were somewhat counter-intuitive; for reasons that are explained below (in the discussion pertaining to sub-banks), there is a slight trend for the performance to decrease as the image size increases.

Data alignment and loop unrolling were issues for unit stride accesses but not for strided accesses. In general, indexed operations behave like strided loads and stores. There is a difference, however, if the indices are loaded from memory, as was the case in this micro-benchmark (they could also for some applications be computed). In this case, there is a stream of unit stride loads for the indices that proceeds in parallel with the indexed load or store stream into the image array. Although data alignment and loop unrolling are therefore potential factors, they do not have significant effects on the performance of the randomized accesses, as the effects of bank and sub-bank conflicts are much more significant. The highest mean peak bandwidth, with the code optimized and the indices aligned, is 0.20 GB/s (25%) for loads and 0.10 GB/s

Lanes	**1**	**2**	**4**	**8**
Peak Bandwidth	**1.6 GB/s**	**3.2 GB/s**	**6.4 GB/s**	**12.8 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.9 (53%)	1.3 (40%)	1.3 (20%)	1.3 (10%)
176 × 144	0.9 (53%)	1.3 (40%)	1.3 (20%)	1.3 (10%)
352 × 240	0.9 (53%)	1.4 (43%)	1.4 (22%)	1.4 (11%)
352 × 288	0.9 (53%)	1.4 (43%)	1.4 (22%)	1.4 (11%)
352 × 480	0.9 (53%)	1.4 (43%)	1.4 (22%)	1.4 (11%)
480 × 480	0.9 (53%)	1.4 (44%)	1.4 (22%)	1.4 (11%)
512 × 384	0.9 (53%)	1.4 (44%)	1.4 (22%)	1.4 (11%)
544 × 480	0.9 (53%)	1.4 (43%)	1.4 (22%)	1.4 (11%)
640 × 480	0.9 (53%)	1.4 (43%)	1.3 (20%)	1.3 (10%)
704 × 480	0.9 (53%)	1.4 (44%)	1.3 (21%)	1.3 (10%)
720 × 400	0.9 (53%)	1.4 (44%)	1.4 (22%)	1.4 (11%)
720 × 480	0.9 (53%)	1.4 (44%)	1.4 (22%)	1.4 (11%)
800 × 600	0.9 (53%)	1.4 (45%)	1.4 (22%)	1.4 (11%)
832 × 624	0.9 (53%)	1.4 (45%)	1.4 (22%)	1.4 (11%)
1024 × 768	0.9 (53%)	1.4 (45%)	1.4 (22%)	1.4 (11%)
1152 × 864	0.9 (53%)	1.4 (43%)	1.3 (20%)	1.3 (10%)
1280 × 720	0.9 (53%)	1.3 (42%)	1.3 (21%)	1.3 (10%)
1280 × 1024	0.9 (53%)	1.3 (42%)	1.3 (21%)	1.3 (10%)
1600 × 1200	0.9 (53%)	1.4 (45%)	1.4 (23%)	1.4 (11%)
1800 × 1440	*1.3 (82%)*	1.4 (45%)	1.4 (22%)	1.4 (11%)
1920 × 1080	0.9 (53%)	1.1 (35%)	1.1 (17%)	1.1 (9%)
1920 × 1200	0.9 (53%)	1.1 (35%)	1.1 (17%)	1.1 (9%)
Median	0.9 (53%)	1.4 (43%)	1.4 (22%)	1.4 (11%)
Mean	0.9 (55%)	1.4 (42%)	1.4 (21%)	1.3 (11%)
Standard deviation	0.10 (6%)	0.09 (3%)	0.09 (1%)	0.10 (1%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.9 (53%)	1.3 (40%)	1.3 (20%)	1.3 (10%)
176 × 144	0.9 (53%)	1.3 (40%)	1.3 (20%)	1.3 (10%)
352 × 240	0.9 (53%)	1.4 (43%)	1.4 (22%)	1.4 (11%)
352 × 288	0.9 (53%)	1.4 (43%)	1.4 (22%)	1.4 (11%)
352 × 480	0.9 (53%)	1.4 (43%)	1.4 (22%)	1.4 (11%)
480 × 480	0.9 (53%)	1.4 (44%)	1.4 (22%)	1.4 (11%)
512 × 384	0.9 (53%)	1.4 (44%)	1.4 (22%)	1.4 (11%)
544 × 480	0.9 (53%)	1.2 (36%)	1.4 (21%)	1.2 (10%)
640 × 480	0.9 (53%)	0.9 (28%)	0.8 (12%)	0.7 (6%)
704 × 480	0.9 (53%)	0.9 (28%)	0.8 (12%)	0.7 (6%)
720 × 400	0.9 (53%)	0.9 (28%)	1.2 (19%)	1.1 (8%)
720 × 480	0.9 (53%)	0.9 (28%)	1.2 (19%)	1.1 (8%)
800 × 600	0.9 (53%)	0.9 (27%)	1.3 (21%)	0.8 (6%)
832 × 624	0.9 (53%)	0.9 (27%)	0.8 (12%)	1.4 (11%)
1024 × 768	0.8 (47%)	0.8 (26%)	0.8 (12%)	0.7 (6%)
1152 × 864	0.9 (53%)	1.0 (32%)	0.9 (14%)	0.9 (7%)
1280 × 720	0.6 (38%)	0.7 (22%)	0.7 (11%)	0.7 (5%)
1280 × 1024	0.6 (38%)	0.7 (22%)	0.7 (11%)	0.7 (5%)
1600 × 1200	0.9 (53%)	0.9 (28%)	0.8 (13%)	1.4 (11%)
1800 × 1440	0.9 (58%)	1.4 (45%)	1.2 (19%)	1.4 (11%)
1920 × 1080	0.5 (31%)	0.6 (19%)	0.6 (10%)	0.6 (5%)
1920 × 1200	0.5 (31%)	0.6 (19%)	0.6 (10%)	0.6 (5%)
Median	0.9 (53%)	0.9 (28%)	1.2 (19%)	1.1 (9%)
Mean	0.8 (50%)	1.0 (32%)	1.1 (17%)	1.1 (8%)
Standard deviation	0.12 (8%)	0.29 (9%)	0.31 (5%)	0.32 (2%)

Figure 6.18: Load and store bandwidth for blocked image access pattern. Inner loop is unrolled 4 times. Data is aligned to the physical width in bits across all of the lanes. Number of lanes is varied. *Italics* is used to highlight a case of special note that is described further in the text.

(12%) for stores. It is interesting to note that, given a sub-bank busy time of 4 cycles for loads and 9 cycles for stores, the mean bandwidth of a randomized access stream is approximately $\frac{1}{4}$ of peak (25%) for loads and $\frac{1}{9}$ of peak (11%) for stores. Although intuition hints that this is a sensible result, a detailed analysis has not been performed to see if such a correlation is always present.

Figure 6.20 shows the results of investigating the performance loss from only having one sub-bank for each bank. In most cases, sub-bank conflicts are a significant contributing factor for the performance of random accesses, and additional sub-banks noticeably helps performance. Compared to 1 sub-bank, the mean performance for loads increases 1.3×, 1.6×, 1.9×, and 2.2× for 2, 4, 8, and 16 sub-banks respectively. The performance for stores increases 1.3×, 1.9×, 2.5×, and 3.2× for 2, 4, 8, and 16 sub-banks respectively. There are a few cases, however, for small images (128 × 96 and 176 × 144) at larger numbers of sub-banks, noted in Figure 6.20 with *italics*, in which the bandwidth levels out at roughly 60% of peak and the addition of more sub-banks does not significantly increase performance.

For these cases, in which the image is small enough and there are a sufficient number of sub-banks, all of the pixels in the entire image map to the same row number spread across all of the sub-banks within all of the banks. An entire image mapping to the highlighted region of Figure 2.2 would be an illustration of this. In such an instance, there are a sufficient number of sub-banks to in general eliminate sub-banks conflicts (perhaps excepting cases in which the misalignment of the image with respect to sub-bank starting addresses causes it to straddle 2 sub-banks), and the addition of sub-banks will not significantly improve performance.

This observation also somewhat explains why, even for 1 sub-bank, smaller images have slightly better performance than larger imaages. Even though the small images do not entirely fit within a single sub-bank, such as the highlighted region in Figure 2.1, for the configurations studied here, the range of the small images only covers a small number of sub-banks. A random sampling of pixels is therefore less likely to encounter sub-bank conflicts than if the range of the image covered many sub-banks, as it does for the larger images. There are less sub-bank conflicts for the smaller images, and a greater percentage of the conflicts are bank conflicts, which have less serious performance consequences than do sub-bank conflicts. The bandwidth achieved for the random accesses pattern for smaller images is therefore somewhat higher than for larger images.

Another factor involved is the merging of addresses. If two 8-bit accesses within a single cycle map to different 64-bit words within the same bank, this causes a bank conflict. However, if those accesses are close enough together that they map to the same 64-bit word, these accesses merge together, and there is no conflict. With a smaller image size, and therefore a smaller range from which random addresses are chosen, there is a slightly higher probability of addresses merging. This merging slightly increases the performance for small images relative to large images, although its effect is less significant than the differences with respect to sub-bank conflicts explained above.

Figure 6.21 shows the effects of scaling the number of lanes from 1 to 8, for 1 to 16 sub-banks. Both the absolute performance and the relative scaling are noticeably affected by the number of sub-banks. At 1 sub-bank the scaling is quite low, as the mean performance increases 1.3×, 1.6×, and 1.7× when scaling from 1 lane to 2, 4, and 8 lanes respectively for loads, and 1.2×, 1.3×, and 1.3× for stores. At 16 sub-banks, the scaling is noticeably improved, though still far from perfect, with the mean increasing 1.8×, 2.7×, and 3.3× when scaling to 2, 4, and 8 lanes for loads, and 1.6×, 2.1×, and 2.5× for stores.

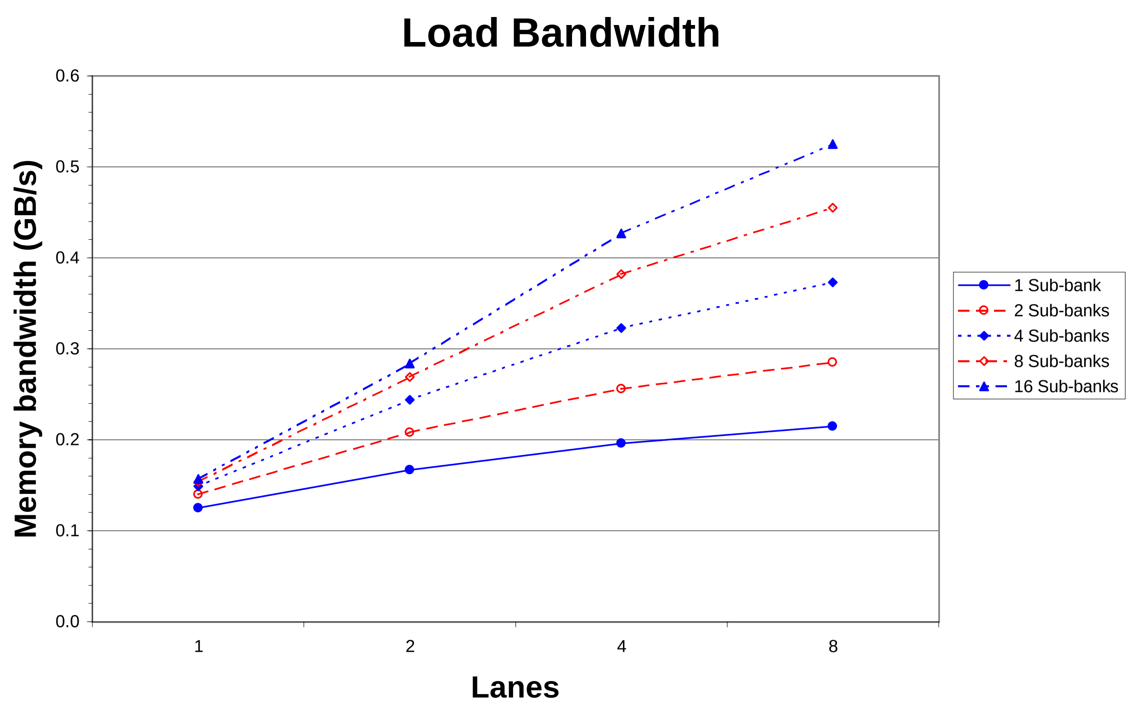
Figure 6.22 shows the effects of scaling up the number of address generation resources from 4 to 16, for 1 to 16 sub-banks. Although the absolute performance is higher with more sub-banks, regardless of how many sub-banks are added, the performance is relatively flat with respect to scaling the number of address generators. Even with 16 sub-banks, the mean bandwidth with 16 address generators only increases 1.1× compared to 4 address generators for loads and 1.1× for stores. Even with a large number of sub-banks, bank conflicts within the memory system prevent fully utilizing the additional address generation resources.

Code Optimized?	**No**	**No**	**Yes**	**Yes**
Data Aligned?	**No**	**Yes**	**No**	**Yes**
Peak Bandwidth	**0.8 GB/s**	**0.8 GB/s**	**0.8 GB/s**	**0.8 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.24 (30%)	0.24 (30%)	0.25 (31%)	0.25 (31%)
176 × 144	0.21 (26%)	0.21 (27%)	0.22 (27%)	0.22 (27%)
352 × 240	0.20 (24%)	0.20 (25%)	0.20 (25%)	0.20 (25%)
352 × 288	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
352 × 480	0.20 (24%)	0.20 (24%)	0.20 (24%)	0.20 (25%)
480 × 480	0.19 (24%)	0.20 (24%)	0.20 (24%)	0.20 (24%)
512 × 384	0.19 (24%)	0.20 (24%)	0.19 (24%)	0.20 (24%)
544 × 480	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
640 × 480	0.19 (23%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
704 × 480	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
720 × 400	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
720 × 480	0.19 (23%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
800 × 600	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
832 × 624	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
1024 × 768	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
1152 × 864	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
1280 × 720	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
1280 × 1024	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
1600 × 1200	0.19 (23%)	0.19 (23%)	0.19 (23%)	0.19 (23%)
1800 × 1440	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
1920 × 1080	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
1920 × 1200	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
Median	0.19 (24%)	0.19 (24%)	0.19 (24%)	0.19 (24%)
Mean	0.19 (24%)	0.20 (24%)	0.20 (24%)	0.20 (25%)
Standard deviation	0.01 (1%)	0.01 (2%)	0.01 (2%)	0.01 (2%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.11 (14%)	0.11 (14%)	0.12 (14%)	0.12 (14%)
176 × 144	0.10 (13%)	0.10 (13%)	0.10 (13%)	0.10 (13%)
352 × 240	0.10 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
352 × 288	0.10 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
352 × 480	0.10 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
480 × 480	0.10 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
512 × 384	0.10 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
544 × 480	0.10 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
640 × 480	0.09 (12%)	0.09 (12%)	0.09 (12%)	0.09 (12%)
704 × 480	0.10 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
720 × 400	0.09 (12%)	0.09 (12%)	0.09 (12%)	0.09 (12%)
720 × 480	0.09 (12%)	0.09 (12%)	0.09 (12%)	0.09 (12%)
800 × 600	0.10 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
832 × 624	0.10 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
1024 × 768	0.09 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
1152 × 864	0.09 (12%)	0.09 (12%)	0.10 (12%)	0.10 (12%)
1280 × 720	0.09 (12%)	0.09 (12%)	0.09 (12%)	0.09 (12%)
1280 × 1024	0.10 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
1600 × 1200	0.09 (11%)	0.09 (11%)	0.09 (11%)	0.09 (12%)
1800 × 1440	0.09 (12%)	0.09 (12%)	0.09 (12%)	0.10 (12%)
1920 × 1080	0.09 (12%)	0.09 (12%)	0.09 (12%)	0.09 (12%)
1920 × 1200	0.09 (12%)	0.09 (12%)	0.09 (12%)	0.10 (12%)
Median	0.10 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
Mean	0.10 (12%)	0.10 (12%)	0.10 (12%)	0.10 (12%)
Standard deviation	0.00 (1%)	0.00 (1%)	0.00 (1%)	0.00 (1%)

Figure 6.19: Load and store bandwidth for randomized image access pattern. Optimized code has the inner loop unrolled twice and utilizes branch delay slots. Unoptimized code is not unrolled and does not use branch delay slots. Alignment of data refers to the index array on 256-bit boundaries.

Sub-banks	**1**	**2**	**4**	**8**	**16**
Peak Bandwidth	**0.8 GB/s**	**0.8 GB/s**	**0.8 GB/s**	**0.8 GB/s**	**0.8 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**				
128 × 96	0.25 (31%)	0.36 (45%)	*0.48 (60%)*	*0.48 (60%)*	*0.48 (60%)*
176 × 144	0.22 (27%)	0.31 (38%)	0.41 (51%)	*0.47 (59%)*	*0.48 (59%)*
352 × 240	0.20 (25%)	0.26 (32%)	0.34 (42%)	0.41 (52%)	0.46 (57%)
352 × 288	0.19 (24%)	0.25 (32%)	0.32 (40%)	0.40 (50%)	0.45 (56%)
352 × 480	0.20 (25%)	0.26 (32%)	0.32 (40%)	0.39 (49%)	0.45 (56%)
480 × 480	0.20 (24%)	0.25 (32%)	0.32 (40%)	0.38 (48%)	0.44 (55%)
512 × 384	0.20 (24%)	0.25 (31%)	0.31 (39%)	0.38 (47%)	0.44 (54%)
544 × 480	0.19 (24%)	0.25 (31%)	0.32 (40%)	0.38 (48%)	0.43 (54%)
640 × 480	0.19 (24%)	0.25 (31%)	0.31 (38%)	0.37 (46%)	0.42 (53%)
704 × 480	0.19 (24%)	0.25 (31%)	0.31 (39%)	0.37 (47%)	0.43 (53%)
720 × 400	0.19 (24%)	0.24 (31%)	0.31 (38%)	0.37 (47%)	0.42 (53%)
720 × 480	0.19 (24%)	0.24 (30%)	0.31 (39%)	0.37 (46%)	0.42 (53%)
800 × 600	0.19 (24%)	0.25 (31%)	0.31 (38%)	0.37 (46%)	0.42 (52%)
832 × 624	0.19 (24%)	0.25 (31%)	0.31 (38%)	0.37 (46%)	0.42 (52%)
1024 × 768	0.19 (24%)	0.25 (31%)	0.31 (38%)	0.36 (45%)	0.41 (51%)
1152 × 864	0.19 (24%)	0.25 (31%)	0.31 (39%)	0.36 (46%)	0.41 (52%)
1280 × 720	0.19 (24%)	0.24 (31%)	0.31 (38%)	0.36 (45%)	0.41 (51%)
1280 × 1024	0.19 (24%)	0.25 (31%)	0.31 (39%)	0.37 (46%)	0.41 (52%)
1600 × 1200	0.19 (23%)	0.24 (30%)	0.30 (37%)	0.35 (44%)	0.40 (50%)
1800 × 1440	0.19 (24%)	0.24 (30%)	0.30 (38%)	0.36 (45%)	0.40 (50%)
1920 × 1080	0.19 (24%)	0.24 (30%)	0.30 (38%)	0.36 (45%)	0.41 (51%)
1920 × 1200	0.19 (24%)	0.25 (31%)	0.30 (38%)	0.36 (44%)	0.40 (50%)
Median	0.19 (24%)	0.25 (31%)	0.31 (39%)	0.37 (46%)	0.42 (53%)
Mean	0.20 (25%)	0.26 (32%)	0.32 (40%)	0.38 (48%)	0.43 (53%)
Standard deviation	0.01 (2%)	0.03 (3%)	0.04 (5%)	0.03 (4%)	0.02 (3%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**				
128 × 96	0.12 (14%)	0.18 (23%)	*0.46 (57%)*	*0.47 (59%)*	*0.48 (60%)*
176 × 144	0.10 (13%)	0.15 (19%)	0.23 (29%)	*0.46 (58%)*	*0.48 (59%)*
352 × 240	0.10 (12%)	0.13 (17%)	0.19 (23%)	0.26 (33%)	0.36 (45%)
352 × 288	0.10 (12%)	0.13 (16%)	0.18 (22%)	0.25 (31%)	0.34 (43%)
352 × 480	0.10 (12%)	0.13 (17%)	0.18 (22%)	0.24 (30%)	0.32 (40%)
480 × 480	0.10 (12%)	0.13 (16%)	0.18 (22%)	0.24 (30%)	0.31 (38%)
512 × 384	0.10 (12%)	0.13 (16%)	0.17 (22%)	0.23 (29%)	0.31 (39%)
544 × 480	0.10 (12%)	0.13 (16%)	0.18 (22%)	0.24 (30%)	0.31 (39%)
640 × 480	0.09 (12%)	0.13 (16%)	0.17 (22%)	0.23 (29%)	0.30 (38%)
704 × 480	0.10 (12%)	0.13 (16%)	0.17 (22%)	0.23 (29%)	0.30 (38%)
720 × 400	0.09 (12%)	0.13 (16%)	0.17 (22%)	0.23 (29%)	0.30 (38%)
720 × 480	0.09 (12%)	0.13 (16%)	0.17 (22%)	0.23 (29%)	0.30 (37%)
800 × 600	0.10 (12%)	0.13 (16%)	0.17 (22%)	0.23 (29%)	0.29 (36%)
832 × 624	0.10 (12%)	0.13 (16%)	0.17 (22%)	0.23 (29%)	0.29 (37%)
1024 × 768	0.10 (12%)	0.13 (16%)	0.17 (21%)	0.22 (28%)	0.28 (35%)
1152 × 864	0.10 (12%)	0.13 (16%)	0.17 (22%)	0.23 (28%)	0.29 (36%)
1280 × 720	0.09 (12%)	0.13 (16%)	0.17 (21%)	0.22 (28%)	0.28 (35%)
1280 × 1024	0.10 (12%)	0.13 (16%)	0.17 (22%)	0.23 (28%)	0.29 (36%)
1600 × 1200	0.09 (12%)	0.12 (16%)	0.17 (21%)	0.22 (27%)	0.28 (35%)
1800 × 1440	0.10 (12%)	0.13 (16%)	0.17 (21%)	0.22 (27%)	0.28 (35%)
1920 × 1080	0.09 (12%)	0.13 (16%)	0.17 (22%)	0.22 (28%)	0.28 (35%)
1920 × 1200	0.10 (12%)	0.13 (16%)	0.17 (21%)	0.22 (27%)	0.28 (35%)
Median	0.10 (12%)	0.13 (16%)	0.17 (22%)	0.23 (29%)	0.30 (37%)
Mean	0.10 (12%)	0.13 (17%)	0.19 (24%)	0.25 (32%)	0.32 (39%)
Standard deviation	0.00 (1%)	0.01 (2%)	0.06 (8%)	0.07 (9%)	0.06 (7%)

Figure 6.20: Load and store bandwidth for randomized image access pattern. Inner loop is unrolled twice and utilizes branch delay slots. Index array is 256-bit aligned. Sub-banks are varied. *Italics* is used to highlight cases of special note that are described further in the text.



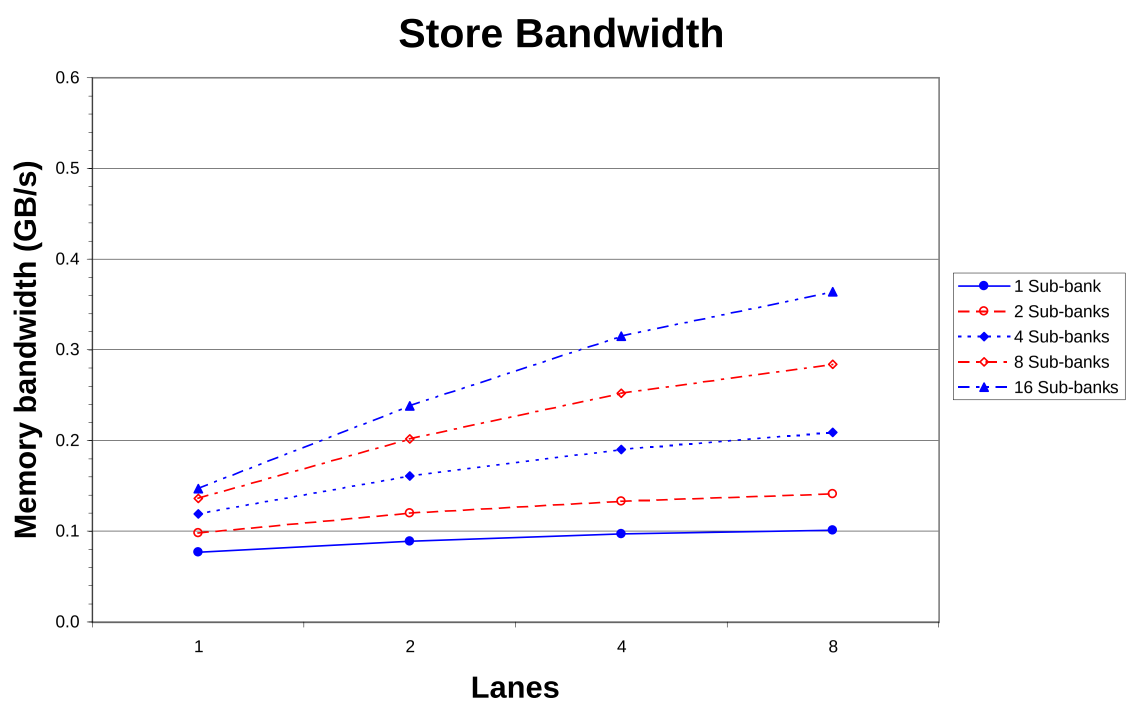
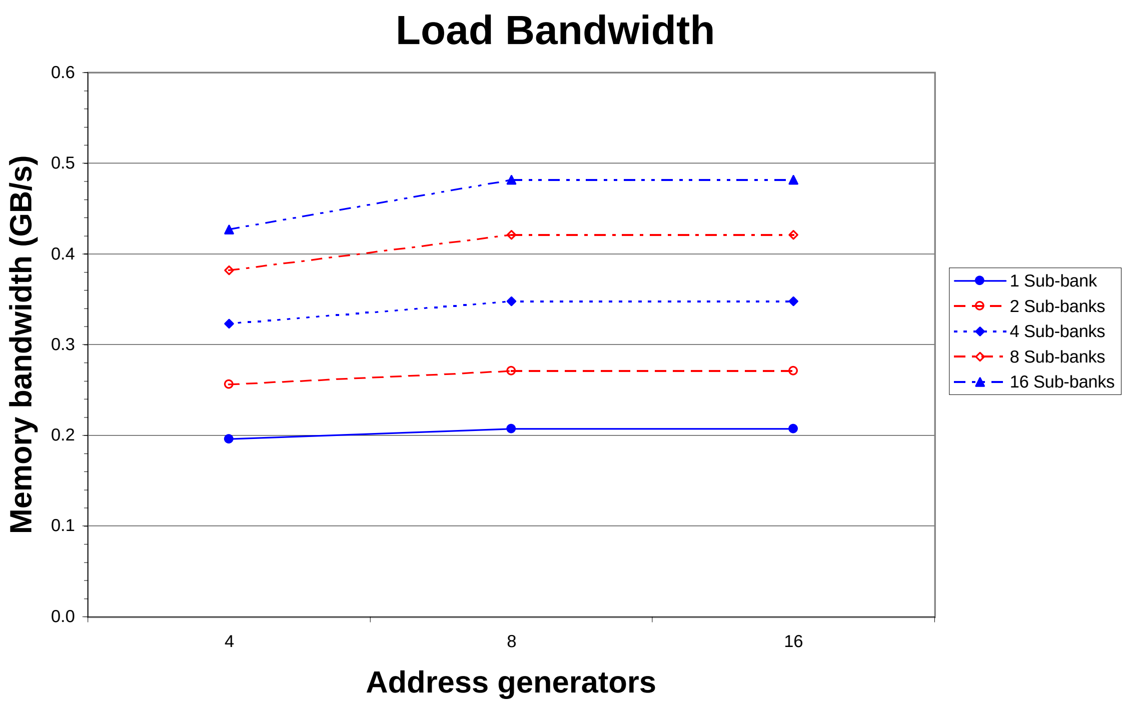


Figure 6.21: Mean load and store bandwidth for randomized image access pattern. Inner loop is unrolled twice and utilizes branch delay slots. Index array is aligned to the physical width in bits across all of the lanes. Lanes and sub-banks are varied. Peak bandwidth is 0.2 GB/s, 0.4 GB/s, 0.8 GB/s and 1.6 GB/s for 1, 2, 4, and 8 lanes respectively. Figures A.12 through A.16 in the Appendix show the data from which these means are derived.



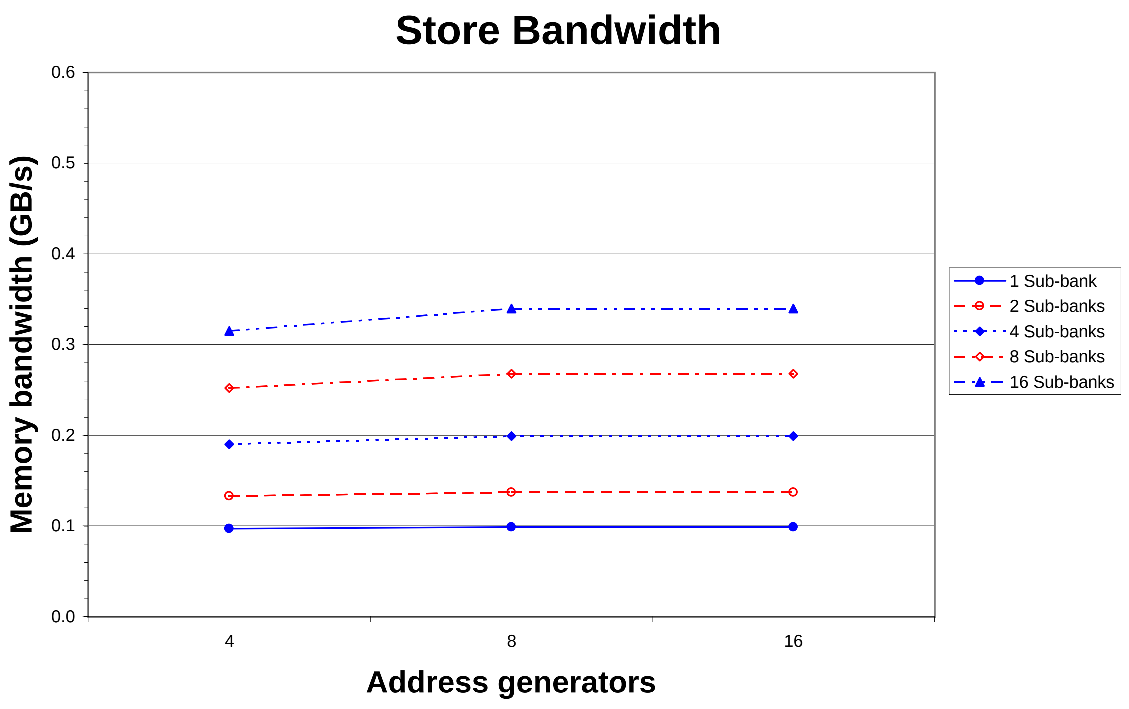


Figure 6.22: Mean load and store bandwidth for randomized image access pattern. Inner loop is unrolled twice and utilizes branch delay slots. Index array is aligned to the physical width in bits across all of the lanes. Address generation resources and sub-banks are varied. Peak bandwidth is 0.8 GB/s, 1.6 GB/s, and 3.2 GB/s for 4, 8, and 16 address generators respectively. Figures A.17 through A.21 in the Appendix show the data from which these means are derived.

Since the indices are loaded from memory, rather than being computed, there are conflicts encountered between the indexed load or store stream accessing the image array and the unit stride loads of the indices. Figure 6.23 investigates an alternative layout and corresponding data placement that tries to address this problem. All of the indices (the unit stride access stream) are placed in one wing, and the entire image array (the indexed access stream) is placed in another wing. Instead of alternating between the wings on subsequent accesses as the default RSBCW layout does, the wing (W) portion of the address decoding is placed at the most significant bit possible, resulting in a layout of WRSBC. Unfortunately, this approach decreases the bandwidth achieved by the indexed stream from a mean of 0.37 GB/s (47%) to 0.32 GB/s (40%) for loads and from 0.37 (46%) to 0.32 GB/s (40%) for stores. Although there is some benefit from the two streams not colliding with one another, this is more than offset by the fact that the indexed access stream has less total banks to choose from (4 instead of 8), resulting in more bank conflicts.

Layout Peak Bandwidth	RSBCW 0.8 GB/s	WRSBC 0.8 GB/s
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**	
128 × 96	0.25 (31%)	0.17 (21%)
176 × 144	0.22 (27%)	0.15 (19%)
352 × 240	0.20 (25%)	0.15 (18%)
352 × 288	0.19 (24%)	0.15 (18%)
352 × 480	0.20 (25%)	0.15 (18%)
480 × 480	0.20 (24%)	0.14 (18%)
512 × 384	0.20 (24%)	0.14 (18%)
544 × 480	0.19 (24%)	0.14 (18%)
640 × 480	0.19 (24%)	0.14 (18%)
704 × 480	0.19 (24%)	0.14 (18%)
720 × 400	0.19 (24%)	0.14 (18%)
720 × 480	0.19 (24%)	0.14 (18%)
800 × 600	0.19 (24%)	0.14 (18%)
832 × 624	0.19 (24%)	0.14 (18%)
1024 × 768	0.19 (24%)	0.14 (18%)
1152 × 864	0.19 (24%)	0.14 (18%)
1280 × 720	0.19 (24%)	0.14 (18%)
1280 × 1024	0.19 (24%)	0.14 (18%)
1600 × 1200	0.19 (23%)	0.14 (18%)
1800 × 1440	0.19 (24%)	0.14 (18%)
1920 × 1080	0.19 (24%)	0.14 (18%)
1920 × 1200	0.19 (24%)	0.14 (18%)
Median	0.19 (24%)	0.14 (18%)
Mean	0.20 (25%)	0.15 (18%)
Standard deviation	0.01 (2%)	0.01 (1%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**	
128 × 96	0.12 (14%)	0.08 (9%)
176 × 144	0.10 (13%)	0.07 (9%)
352 × 240	0.10 (12%)	0.07 (9%)
352 × 288	0.10 (12%)	0.07 (9%)
352 × 480	0.10 (12%)	0.07 (9%)
480 × 480	0.10 (12%)	0.07 (9%)
512 × 384	0.10 (12%)	0.07 (9%)
544 × 480	0.10 (12%)	0.07 (9%)
640 × 480	0.09 (12%)	0.07 (9%)
704 × 480	0.10 (12%)	0.07 (8%)
720 × 400	0.09 (12%)	0.07 (9%)
720 × 480	0.09 (12%)	0.07 (9%)
800 × 600	0.10 (12%)	0.07 (8%)
832 × 624	0.10 (12%)	0.07 (8%)
1024 × 768	0.10 (12%)	0.07 (9%)
1152 × 864	0.10 (12%)	0.07 (9%)
1280 × 720	0.09 (12%)	0.07 (9%)
1280 × 1024	0.10 (12%)	0.07 (9%)
1600 × 1200	0.09 (12%)	0.07 (8%)
1800 × 1440	0.10 (12%)	0.07 (9%)
1920 × 1080	0.09 (12%)	0.07 (9%)
1920 × 1200	0.10 (12%)	0.07 (9%)
Median	0.10 (12%)	0.07 (9%)
Mean	0.10 (12%)	0.07 (9%)
Standard deviation	0.00 (1%)	0.00 (0%)

Figure 6.23: Load and store bandwidth for randomized image access pattern. Inner loop is unrolled twice and utilizes branch delay slots. Index array is 256-bit aligned. Alternative layout is compared to default

Section 7

Conclusions and Future Work

Figure 7.1 shows the mean load and store bandwidths for the default hardware configuration for the best possible software configuration (with the inner loop unrolled and the data aligned where applicable) for each of the image access patterns studied. Clearly it is easiest to achieve peak performance with the unit stride accesses of the horizontal image access pattern. The strided accesses of the vertical image access pattern average slightly less than one half of peak performance for loads and slightly less than one quarter of peak performance for stores. The indexed accesses of the random image access pattern average approximate one quarter of peak performance for loads and one ninth of peak performance for stores.

Averages can be deceiving, however, as there is high variability in the vertical image access pattern. For the default hardware configuration, the load performance ranges from 0.17 GB/s (21%) to 0.60 GB/s (75%), and the store performance ranges from 0.08 GB/s (10%) to 0.37 GB/s (46%). And one particular data point (1800 × 1440) has nearly 50% greater load performance than the mean for one particular case (8 × 8 blocked image access pattern with 1 lane) simply because the image width is not divisible by 16.

Loop unrolling and data alignment were important in some cases for maximizing performance. Loop unrolling was necessary when there was insufficient issue bandwidth to keep one or both memory units busy, in the horizontal and blocked image access patterns. Data alignment was only significant when there was sufficient issue bandwidth to keep the vector memory unit(s) busy. In one such case, for the horizontal image access pattern, it meant the difference between 80% and 100% of peak performance.

Access Pattern	**Bandwidth in GB/s (Percentage of Peak)**				
		Load		**Store**	
	Peak	**Mean**	**Std dev**	**Mean**	**Std dev**
Horizontal	6.4	6.4 (100%)	0.01 (0%)	6.4 (100%)	0.01 (0%)
Vertical	0.8	0.38 (47%)	0.16 (19%)	0.19 (24%)	0.09 (11%)
8 × 8 Blocked	6.4	1.4 (21%)	0.09 (1%)	1.1 (17%)	0.31 (5%)
Random	0.8	0.20 (25%)	0.01 (2%)	0.10 (12%)	0.00 (1%)

Figure 7.1: Summary of results. Mean load and store bandwidths for the micro-benchmarks of 8-bit pixel data accessed in a 16-bit `vpw` for the default hardware configuration for the best possible software configuration (inner loop unrolled and data aligned where applicable).

An alternative layout (RCSBW instead of the default RSBCW) and an address hashing scheme (the addition of XORing on the bank selection bits) were evaluated to address bank conflicts experienced by the vertical image access pattern. Although levels of XORing significantly benefitted some cases and reduced the variance, on average the performance was worse with XORing than without. The alternative layout benefitted some cases but considerably reduced the performance for others. The variance was larger with the alternative layout, although on average it helped marginally. The benefits were greater for loads than for stores, and the benefits diminished with higher XOR levels. Since there is a significant potential for decreasing performance in certain situations, in addition to the potential for increasing performance in others, it is suggested that these features, if incorporated into VIRAM-1, be selectable by the programmer on a per-process basis.

Alternative data layouts and address hashing schemes are hardware solutions to attacking the variablility of strided performance. This problem could also be addressed with software solutions by either the programmer or the compiler, like padding arrays to odd row widths or rearranging data in memory to alter the access pattern. In the most extreme case, any arbitrary address could map to any arbitrary placement in physical memory. What is important is finding data layout and addressing schemes (whether implemented in software or hardware) that maximize average performance and are not overly complicated to implement. There will likely always be some pathological cases where the performance is poor for some fixed solution. The hope is that the number of such cases can be mimized, and with a flexible solution such cases could hopefully be avoided altogether if the access pattern is known in advance.

Within the memory system, both sub-bank and bank conflicts contributed to limitations in performance. Due to the extra sub-bank busy time for stores, in instances in which sub-bank conflicts were significant, the store performance was considerably lower than the load performance. VIRAM-1 only has 1 sub-bank per each memory bank due to limitations of the DRAM macro we are receiving from an industrial partner. Increasing the number of sub-banks per bank helped considerably in some cases, with the majority of the benefits being realized in most cases when increasing to 2 and 4 sub-banks. However, in other cases bank conflicts were a more significant factor, and increasing the number of sub-banks helped little. The number of banks in the memory system is limited because there is a significant amount of overhead associated with each independent bank, and increasing the number of banks has significant area costs.

The memory system was often a limiting factor in the ability of the vector unit to effectively scale. With only 1 sub-bank, the scaling in performance as the number of lanes was scaled from 1 to 8 and as the number of address generators was (separately) scaled from 4 to 16 was far from perfect for all cases except the horizontal image access pattern. The scaling improved as the number of sub-banks was increased for those cases in which sub-bank conflicts were a significant factor. Even for those cases, however, scaling was still limited by bank conflicts. An observation is that there is no point in generating more addresses to send to the memory system if the memory system does not have the ability to process those addresses.

In several instances short vectors were a limitation to performance. Short vectors, which result in shorter chimes, put more pressure on the available vector issue bandwidth. The simplified pipelined control of VIRAM-1 means that stalls in one instruction can adversely affect instructions even in other VFUs, and this becomes more of an issue with shorter chimes. The chime length is shortened when the number of lanes is increased, meaning that short vector problems are another potential limitation to the scaling of performance as the number of lanes is scaled. The best possible bandwidth that could have been achieved for the 8×8 blocked image access pattern was only 25% of peak due to short vectors. Short vectors would be less of a problem, and the resulting bandwidth would be higher, for 16×16 blocks, which are also used by MPEG codecs in addition to the 8×8 blocks studied here.

The preceeding discussion has raised several issues. Sometimes bank conflicts are the main issue within the memory system, sometimes sub-bank conflicts are. Sometimes loop unrolling and data alignment are needed, sometimes they are not. Sometimes an alternative layout would help, sometimes it would hurt. Sometimes a particular address hashing scheme would help, sometimes it would hurt. This study initially set out to answer the question, "What is the performance of the VIRAM-1 memory system?", and even with simple micro-benchmarks the best answer seems to be "It depends."

The memory performance depends on the type of access pattern. Unit stride accesses have the highest performance, and non-unit stride accesses have significantly lower performance. Compared to conventional, cache based machines, however, the absolute performances, even for non-unit strides, are reasonably good. For instance, the highest figure amongst *any* Intel x86 based PC listed in the current (July 1999) results for the memory to memory copy STREAM benchmark [20] is 304.0 MB/s. Note that the clock rate for this machine is 400 MHz, twice that of VIRAM-1, and that this benchmark is one for which VIRAM is capable of achieving peak performance, due to the large unit stride access streams.

For strided accesses, the memory performance depends largely on the stride and has significant variability. The behavior of certain strides is much better than the mean, close to peak performance, while other strides are pathological cases with poor performance much worse than the mean. If the data access patterns are predictable, programmers or compilers can optimize for performance with either software solutions, such as padding rows with extra elements, or hardware solutions, such as choosing an appropriate address mapping. However, the data access patterns for an application are not always simple and known at compile time. Searching for and testing ideas that could both increase the average performance for non-unit stride accesses and decrease the variance, including addressing schemes more complex than a simple linear mapping of the data layout or a simple hashing of the address using a few XOR levels, is future work. This includes investigating ideas have been proposed and/or utilized in the past and determining if they might be applicable to VIRAM.

Recent design decisions for VIRAM-1 have included dropping the second memory unit from the implementation and investigating the possibility of decoupled loads and stores, which becomes feasible to implement once there is only a single memory unit.

Having only one memory unit available will halve the peak unit stride performance from its current value (from 6.4 GB/s to 3.2 GB/s for the image accesses studied here) but will not affect the peak performance of strided and indexed operations. These will stay at 0.8 GB/s since they previously had only been able to utilize a single memory unit. The absolute performance of the horizontal image access pattern will be halved, but it should still be easy to maintain 100% of peak for this micro-benchmark. The vertical image access pattern should be unaffected, since it is entirely constrained to a single memory unit. Likewise, short vectors prevented the unit stride accesses of the 8×8 blocked pattern from utilizing both memory units with the default configuration, implying that its performance will also be unaffected. The randomized image access pattern could be somewhat adversely affected, since the indices were loaded from memory with a unit stride access stream that will have additional competition for resources with the indexed access stream in the presence of only a single vector memory unit.

Although the measured bandwidths for all but the horizontal image access pattern are well below peak, a real application does not entirely consist of loads and stores, and the hope is that with a sufficiently high computation to memory ratio and proper scheduling that the memory bandwidths observed here will be sufficient and that an application can keep all of the functional units busy. A problem with this assumption is that most any stall in a memory functional unit causes stalls in the arithmetic functional units as well, due to the simplified pipeline control of VIRAM-1.

Decoupled loads and stores can address this problem. The lower the performance of non-unit stride accesses is from peak, the more potential gain there is from decoupled loads and stores. The implementation being considered for decoupled loads and/or stores would only involve non-unit stride accesses, since a realistic implementation for VIRAM-1 will require disabling chaining, which is essential for maintaining peak performance for unit stride accesses.

Future work includes quantitatively evaluating the impact of dropping the second memory unit and decoupled loads and stores on performance.

The current simulation tools include a simplified, single-issue model of the scalar core. More accurate results, especially for studies beyond simple micro-benchmarks, will be possible once a detailed, dual-issue model of the scalar core to be used in VIRAM-1 is integrated with the existing model of the vector unit, and once scalar cache invalidation traffic is included as well.

Finally, in order to determine how much the issues outlined above ultimately matter, we need to study the performance of more extensive kernels and real applications, and not simply micro-benchmarks. This is ongoing and future work.

Appendix A

Graph Data

This appendix contains the complete data for which the graphs in Figures 6.15, 6.16, 6.21, and 6.22 are derived. Figure A.1 summarizes the relationships between the data in this appendix and the corresponding graphs in the main text.

Figure	**Access Pattern**	**Plotted as**		**Sub-banks**	**Variable**	**Notes**
		Figure	**Line**			
A.2	Vertical	6.15	1	1	1-8 lanes	RSBCW layout, 0 XOR levels
A.3			2	2		
A.4			3	4		
A.5			4	8		
A.6			5	16		
A.7	Vertical	6.16	1	1	4-16 address generators	RSBCW layout, 0 XOR levels
A.8			2	2		
A.9			3	4		
A.10			4	8		
A.11			5	16		
A.12	Random	6.21	1	1	1-8 lanes	Code optimized, Data aligned
A.13			2	2		
A.14			3	4		
A.15			4	8		
A.16			5	16		
A.17	Random	6.22	1	1	4-16 address generators	Code optimized, Data aligned
A.18			2	2		
A.19			3	4		
A.20			4	8		
A.21			5	16		

Figure A.1: Data to graph mapping. Each figure in the appendix contains tables (for load and store) whose mean data corresponds to a a single line on a graph (one for load, one for store) in the main text.

Lanes	**1**	**2**	**4**	**8**
Peak Bandwidth	**0.2 GB/s**	**0.4 GB/s**	**0.8 GB/s**	**1.6 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.20 (98%)	0.22 (54%)	0.26 (33%)	0.26 (16%)
176 × 144	0.20 (100%)	0.39 (97%)	0.41 (51%)	0.50 (31%)
352 × 240	0.20 (100%)	0.30 (76%)	0.60 (75%)	0.53 (33%)
352 × 288	0.20 (100%)	0.30 (76%)	0.60 (75%)	0.53 (33%)
352 × 480	0.20 (100%)	0.30 (76%)	0.60 (75%)	0.53 (33%)
480 × 480	0.20 (100%)	0.37 (93%)	0.53 (66%)	0.44 (27%)
512 × 384	0.20 (100%)	0.40 (100%)	0.40 (50%)	0.40 (25%)
544 × 480	0.20 (100%)	0.35 (88%)	0.53 (66%)	0.38 (24%)
640 × 480	0.20 (100%)	0.31 (78%)	0.32 (40%)	0.32 (20%)
704 × 480	0.20 (100%)	0.30 (76%)	0.31 (38%)	0.32 (20%)
720 × 400	0.20 (100%)	0.31 (77%)	0.55 (69%)	0.38 (24%)
720 × 480	0.20 (100%)	0.31 (77%)	0.55 (69%)	0.38 (23%)
800 × 600	0.20 (100%)	0.24 (61%)	0.48 (60%)	0.33 (21%)
832 × 624	0.20 (100%)	0.24 (61%)	0.25 (31%)	0.51 (32%)
1024 × 768	0.20 (100%)	0.20 (50%)	0.20 (25%)	0.20 (12%)
1152 × 864	0.20 (100%)	0.32 (80%)	0.34 (42%)	0.34 (21%)
1280 × 720	0.16 (80%)	0.17 (44%)	0.18 (22%)	0.19 (12%)
1280 × 1024	0.16 (80%)	0.17 (44%)	0.18 (22%)	0.19 (12%)
1600 × 1200	0.20 (100%)	0.27 (66%)	0.32 (40%)	0.65 (41%)
1800 × 1440	0.19 (94%)	0.38 (95%)	0.39 (48%)	0.49 (30%)
1920 × 1080	0.13 (66%)	0.16 (40%)	0.17 (21%)	0.18 (11%)
1920 × 1200	0.13 (66%)	0.16 (40%)	0.17 (21%)	0.18 (11%)
Median	0.20 (100%)	0.30 (76%)	0.36 (45%)	0.38 (23%)
Mean	0.19 (95%)	0.28 (70%)	0.38 (47%)	0.37 (23%)
Standard deviation	0.02 (11%)	0.08 (19%)	0.16 (19%)	0.14 (9%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.19 (93%)	0.20 (51%)	0.24 (30%)	0.24 (15%)
176 × 144	0.19 (96%)	0.36 (90%)	0.36 (46%)	0.40 (25%)
352 × 240	0.20 (99%)	0.23 (58%)	0.27 (34%)	0.25 (16%)
352 × 288	0.20 (100%)	0.23 (59%)	0.27 (34%)	0.25 (16%)
352 × 480	0.20 (99%)	0.23 (58%)	0.27 (34%)	0.25 (16%)
480 × 480	0.19 (93%)	0.19 (46%)	0.28 (36%)	0.20 (12%)
512 × 384	0.18 (89%)	0.18 (44%)	0.18 (22%)	0.18 (11%)
544 × 480	0.17 (84%)	0.17 (42%)	0.28 (36%)	0.17 (11%)
640 × 480	0.14 (71%)	0.14 (36%)	0.15 (18%)	0.15 (9%)
704 × 480	0.14 (68%)	0.14 (35%)	0.14 (18%)	0.14 (9%)
720 × 400	0.13 (66%)	0.14 (36%)	0.26 (33%)	0.17 (11%)
720 × 480	0.13 (66%)	0.14 (36%)	0.26 (33%)	0.17 (11%)
800 × 600	0.11 (57%)	0.11 (28%)	0.23 (28%)	0.16 (10%)
832 × 624	0.11 (57%)	0.11 (28%)	0.11 (14%)	0.23 (14%)
1024 × 768	0.09 (44%)	0.09 (22%)	0.09 (11%)	0.09 (6%)
1152 × 864	0.16 (80%)	0.16 (40%)	0.16 (20%)	0.16 (10%)
1280 × 720	0.08 (40%)	0.08 (21%)	0.08 (10%)	0.09 (5%)
1280 × 1024	0.08 (40%)	0.08 (21%)	0.08 (10%)	0.09 (5%)
1600 × 1200	0.13 (66%)	0.14 (36%)	0.16 (20%)	0.32 (20%)
1800 × 1440	0.16 (82%)	0.22 (54%)	0.18 (23%)	0.23 (14%)
1920 × 1080	0.07 (36%)	0.08 (20%)	0.08 (10%)	0.09 (5%)
1920 × 1200	0.07 (36%)	0.08 (20%)	0.08 (10%)	0.09 (5%)
Median	0.14 (70%)	0.14 (36%)	0.18 (22%)	0.17 (11%)
Mean	0.14 (71%)	0.16 (40%)	0.19 (24%)	0.19 (12%)
Standard deviation	0.04 (22%)	0.07 (17%)	0.09 (11%)	0.08 (5%)

Figure A.2: Load and store bandwidth for vertical image access pattern. The mean data for these cases are the first lines on the graphs in Figure 6.15. Layout is fixed at RSBCW. XORing is fixed at 0 levels. Sub-banks are fixed at 1. Lanes are varied.

Lanes **Peak Bandwidth**	**1** **0.2 GB/s**	**2** **0.4 GB/s**	**4** **0.8 GB/s**	**8** **1.6 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.20 (100%)	0.22 (55%)	0.27 (33%)	0.27 (17%)
176 × 144	0.20 (100%)	0.39 (97%)	0.41 (51%)	0.50 (31%)
352 × 240	0.20 (100%)	0.30 (76%)	0.77 (96%)	0.80 (50%)
352 × 288	0.20 (100%)	0.30 (76%)	0.78 (97%)	0.81 (51%)
352 × 480	0.20 (100%)	0.30 (76%)	0.79 (99%)	0.83 (52%)
480 × 480	0.20 (100%)	0.37 (93%)	0.78 (98%)	0.85 (53%)
512 × 384	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.80 (50%)
544 × 480	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.71 (45%)
640 × 480	0.20 (100%)	0.40 (100%)	0.61 (77%)	0.56 (35%)
704 × 480	0.20 (100%)	0.40 (99%)	0.57 (72%)	0.54 (34%)
720 × 400	0.20 (100%)	0.39 (98%)	0.56 (70%)	0.56 (35%)
720 × 480	0.20 (100%)	0.40 (99%)	0.56 (70%)	0.56 (35%)
800 × 600	0.20 (100%)	0.40 (100%)	0.48 (60%)	0.83 (52%)
832 × 624	0.20 (100%)	0.40 (100%)	0.48 (60%)	0.51 (32%)
1024 × 768	0.20 (100%)	0.40 (100%)	0.40 (50%)	0.40 (25%)
1152 × 864	0.20 (100%)	0.32 (80%)	0.34 (42%)	0.34 (21%)
1280 × 720	0.20 (100%)	0.40 (99%)	0.47 (59%)	0.56 (35%)
1280 × 1024	0.20 (100%)	0.40 (100%)	0.48 (60%)	0.57 (36%)
1600 × 1200	0.20 (100%)	0.27 (66%)	0.32 (40%)	0.65 (41%)
1800 × 1440	0.20 (100%)	0.40 (100%)	0.47 (59%)	0.75 (47%)
1920 × 1080	0.20 (100%)	0.40 (100%)	0.44 (55%)	0.55 (34%)
1920 × 1200	0.20 (100%)	0.40 (100%)	0.44 (55%)	0.55 (34%)
Median	0.20 (100%)	0.40 (99%)	0.48 (60%)	0.56 (35%)
Mean	0.20 (100%)	0.37 (92%)	0.55 (68%)	0.61 (38%)
Standard deviation	0.00 (0%)	0.05 (13%)	0.17 (21%)	0.17 (10%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.20 (100%)	0.22 (55%)	0.27 (33%)	0.27 (17%)
176 × 144	0.19 (96%)	0.36 (90%)	0.38 (47%)	0.46 (28%)
352 × 240	0.20 (99%)	0.29 (74%)	0.48 (61%)	0.47 (30%)
352 × 288	0.20 (100%)	0.30 (74%)	0.49 (61%)	0.48 (30%)
352 × 480	0.20 (100%)	0.30 (76%)	0.50 (63%)	0.49 (31%)
480 × 480	0.20 (99%)	0.35 (87%)	0.48 (60%)	0.39 (24%)
512 × 384	0.20 (100%)	0.36 (89%)	0.36 (44%)	0.36 (22%)
544 × 480	0.20 (100%)	0.32 (80%)	0.63 (78%)	0.34 (21%)
640 × 480	0.20 (100%)	0.28 (70%)	0.29 (36%)	0.29 (18%)
704 × 480	0.20 (99%)	0.26 (64%)	0.26 (33%)	0.28 (18%)
720 × 400	0.20 (98%)	0.29 (71%)	0.27 (33%)	0.27 (17%)
720 × 480	0.20 (98%)	0.29 (71%)	0.27 (33%)	0.27 (17%)
800 × 600	0.20 (100%)	0.23 (57%)	0.23 (28%)	0.45 (28%)
832 × 624	0.20 (100%)	0.23 (57%)	0.23 (28%)	0.23 (14%)
1024 × 768	0.18 (89%)	0.18 (44%)	0.18 (22%)	0.18 (11%)
1152 × 864	0.16 (80%)	0.16 (40%)	0.16 (20%)	0.16 (10%)
1280 × 720	0.20 (100%)	0.29 (73%)	0.29 (36%)	0.29 (18%)
1280 × 1024	0.20 (100%)	0.30 (74%)	0.29 (37%)	0.29 (18%)
1600 × 1200	0.13 (66%)	0.14 (36%)	0.16 (20%)	0.32 (20%)
1800 × 1440	0.20 (100%)	0.26 (64%)	0.22 (28%)	0.40 (25%)
1920 × 1080	0.20 (100%)	0.30 (76%)	0.30 (38%)	0.30 (19%)
1920 × 1200	0.20 (100%)	0.30 (76%)	0.30 (38%)	0.30 (19%)
Median	0.20 (100%)	0.29 (72%)	0.29 (36%)	0.30 (19%)
Mean	0.19 (96%)	0.27 (68%)	0.32 (40%)	0.33 (21%)
Standard deviation	0.02 (8%)	0.06 (15%)	0.12 (16%)	0.09 (6%)

Figure A.3: Load and store bandwidth for vertical image access pattern. The mean data for these cases are the second lines on the graphs in Figure 6.15. Layout is fixed at RSBCW. XORing is fixed at 0 levels. Sub-banks are fixed at 2. Lanes are varied.

Lanes **Peak Bandwidth**	**1** **0.2 GB/s**	**2** **0.4 GB/s**	**4** **0.8 GB/s**	**8** **1.6 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.20 (100%)	0.22 (55%)	0.27 (33%)	0.27 (17%)
176 × 144	0.20 (100%)	0.40 (100%)	0.42 (52%)	0.52 (32%)
352 × 240	0.20 (100%)	0.30 (76%)	0.77 (96%)	0.82 (51%)
352 × 288	0.20 (100%)	0.30 (76%)	0.78 (97%)	0.82 (52%)
352 × 480	0.20 (100%)	0.30 (76%)	0.79 (99%)	0.85 (53%)
480 × 480	0.20 (100%)	0.37 (93%)	0.78 (98%)	1.57 (98%)
512 × 384	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.60 (100%)
544 × 480	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.29 (80%)
640 × 480	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.80 (50%)
704 × 480	0.20 (100%)	0.40 (100%)	0.79 (99%)	0.79 (50%)
720 × 400	0.20 (100%)	0.40 (100%)	0.79 (99%)	0.98 (61%)
720 × 480	0.20 (100%)	0.40 (99%)	0.80 (100%)	0.98 (62%)
800 × 600	0.20 (100%)	0.40 (100%)	0.79 (98%)	0.83 (52%)
832 × 624	0.20 (100%)	0.40 (100%)	0.80 (99%)	0.95 (60%)
1024 × 768	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.80 (50%)
1152 × 864	0.20 (100%)	0.40 (100%)	0.67 (83%)	0.60 (37%)
1280 × 720	0.20 (100%)	0.40 (99%)	0.47 (59%)	0.56 (35%)
1280 × 1024	0.20 (100%)	0.40 (100%)	0.48 (60%)	0.57 (36%)
1600 × 1200	0.20 (100%)	0.40 (100%)	0.55 (69%)	0.65 (41%)
1800 × 1440	0.20 (100%)	0.40 (100%)	0.47 (59%)	0.75 (47%)
1920 × 1080	0.20 (100%)	0.40 (100%)	0.45 (56%)	0.57 (36%)
1920 × 1200	0.20 (100%)	0.40 (100%)	0.45 (57%)	0.58 (36%)
Median	0.20 (100%)	0.40 (100%)	0.78 (98%)	0.80 (50%)
Mean	0.20 (100%)	0.38 (94%)	0.66 (82%)	0.83 (52%)
Standard deviation	0.00 (0%)	0.05 (12%)	0.17 (22%)	0.32 (20%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.20 (100%)	0.22 (55%)	0.27 (33%)	0.27 (17%)
176 × 144	0.20 (100%)	0.40 (100%)	0.42 (52%)	0.52 (32%)
352 × 240	0.20 (99%)	0.29 (74%)	0.71 (89%)	0.74 (46%)
352 × 288	0.20 (100%)	0.30 (74%)	0.73 (91%)	0.76 (47%)
352 × 480	0.20 (100%)	0.30 (76%)	0.76 (95%)	0.80 (50%)
480 × 480	0.20 (99%)	0.37 (92%)	0.69 (86%)	0.86 (54%)
512 × 384	0.20 (100%)	0.40 (100%)	0.71 (89%)	0.71 (44%)
544 × 480	0.20 (100%)	0.40 (100%)	0.64 (80%)	1.10 (69%)
640 × 480	0.20 (100%)	0.40 (100%)	0.55 (68%)	0.55 (34%)
704 × 480	0.20 (100%)	0.40 (99%)	0.47 (59%)	0.76 (48%)
720 × 400	0.20 (100%)	0.40 (100%)	0.51 (63%)	0.52 (33%)
720 × 480	0.20 (98%)	0.39 (97%)	0.52 (65%)	0.52 (33%)
800 × 600	0.20 (100%)	0.39 (98%)	0.47 (59%)	0.45 (28%)
832 × 624	0.20 (100%)	0.40 (99%)	0.45 (57%)	0.46 (29%)
1024 × 768	0.20 (100%)	0.36 (89%)	0.36 (44%)	0.36 (22%)
1152 × 864	0.20 (100%)	0.32 (80%)	0.33 (41%)	0.31 (19%)
1280 × 720	0.20 (100%)	0.29 (73%)	0.29 (36%)	0.29 (18%)
1280 × 1024	0.20 (100%)	0.30 (74%)	0.29 (37%)	0.29 (18%)
1600 × 1200	0.20 (100%)	0.27 (66%)	0.29 (37%)	0.32 (20%)
1800 × 1440	0.20 (100%)	0.26 (64%)	0.22 (28%)	0.40 (25%)
1920 × 1080	0.20 (100%)	0.36 (90%)	0.36 (45%)	0.37 (23%)
1920 × 1200	0.20 (100%)	0.36 (90%)	0.36 (46%)	0.37 (23%)
Median	0.20 (100%)	0.36 (90%)	0.46 (58%)	0.49 (31%)
Mean	0.20 (100%)	0.34 (86%)	0.47 (59%)	0.53 (33%)
Standard deviation	0.00 (0%)	0.06 (14%)	0.17 (21%)	0.23 (14%)

Figure A.4: Load and store bandwidth for vertical image access pattern. The mean data for these cases are the third lines on the graphs in Figure 6.15. Layout is fixed at RSBCW. XORing is fixed at 0 levels. Sub-banks are fixed at 4. Lanes are varied.

Lanes Peak Bandwidth	1 0.2 GB/s	2 0.4 GB/s	4 0.8 GB/s	8 1.6 GB/s
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.20 (100%)	0.22 (55%)	0.27 (33%)	0.27 (17%)
176 × 144	0.20 (100%)	0.40 (100%)	0.42 (52%)	0.52 (32%)
352 × 240	0.20 (100%)	0.30 (76%)	0.80 (100%)	0.86 (54%)
352 × 288	0.20 (100%)	0.30 (76%)	0.78 (97%)	0.82 (52%)
352 × 480	0.20 (100%)	0.30 (76%)	0.79 (99%)	0.85 (53%)
480 × 480	0.20 (100%)	0.37 (93%)	0.78 (98%)	1.57 (98%)
512 × 384	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.60 (100%)
544 × 480	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.29 (81%)
640 × 480	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.80 (50%)
704 × 480	0.20 (100%)	0.40 (100%)	0.79 (99%)	0.79 (50%)
720 × 400	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.09 (68%)
720 × 480	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.09 (68%)
800 × 600	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.88 (55%)
832 × 624	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.60 (100%)
1024 × 768	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.80 (50%)
1152 × 864	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.89 (56%)
1280 × 720	0.20 (100%)	0.40 (99%)	0.51 (64%)	0.62 (39%)
1280 × 1024	0.20 (100%)	0.40 (100%)	0.52 (64%)	0.63 (39%)
1600 × 1200	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.57 (98%)
1800 × 1440	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.03 (64%)
1920 × 1080	0.20 (100%)	0.40 (100%)	0.45 (56%)	0.57 (36%)
1920 × 1200	0.20 (100%)	0.40 (100%)	0.45 (57%)	0.58 (36%)
Median	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.86 (53%)
Mean	0.20 (100%)	0.38 (94%)	0.70 (87%)	0.94 (59%)
Standard deviation	0.00 (0%)	0.05 (12%)	0.17 (21%)	0.38 (24%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.20 (100%)	0.22 (55%)	0.27 (33%)	0.27 (17%)
176 × 144	0.20 (100%)	0.40 (100%)	0.42 (52%)	0.52 (32%)
352 × 240	0.20 (100%)	0.30 (76%)	0.80 (100%)	0.86 (54%)
352 × 288	0.20 (100%)	0.30 (74%)	0.73 (91%)	0.77 (48%)
352 × 480	0.20 (100%)	0.30 (76%)	0.76 (95%)	0.82 (51%)
480 × 480	0.20 (99%)	0.37 (92%)	0.75 (94%)	1.39 (87%)
512 × 384	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.42 (89%)
544 × 480	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.17 (73%)
640 × 480	0.20 (100%)	0.40 (100%)	0.77 (96%)	0.77 (48%)
704 × 480	0.20 (100%)	0.40 (99%)	0.76 (95%)	0.76 (48%)
720 × 400	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.93 (58%)
720 × 480	0.20 (100%)	0.40 (100%)	0.78 (98%)	0.89 (56%)
800 × 600	0.20 (100%)	0.40 (100%)	0.79 (98%)	0.81 (51%)
832 × 624	0.20 (100%)	0.40 (100%)	0.80 (99%)	1.60 (100%)
1024 × 768	0.20 (100%)	0.40 (100%)	0.71 (89%)	0.71 (44%)
1152 × 864	0.20 (100%)	0.40 (99%)	0.78 (97%)	0.84 (53%)
1280 × 720	0.20 (100%)	0.39 (98%)	0.50 (63%)	0.61 (38%)
1280 × 1024	0.20 (100%)	0.40 (100%)	0.52 (64%)	0.63 (39%)
1600 × 1200	0.20 (100%)	0.40 (100%)	0.79 (98%)	1.52 (95%)
1800 × 1440	0.20 (100%)	0.40 (100%)	0.51 (64%)	0.56 (35%)
1920 × 1080	0.20 (100%)	0.36 (90%)	0.36 (45%)	0.37 (23%)
1920 × 1200	0.20 (100%)	0.36 (90%)	0.37 (46%)	0.37 (23%)
Median	0.20 (100%)	0.40 (100%)	0.76 (95%)	0.79 (49%)
Mean	0.20 (100%)	0.37 (93%)	0.66 (83%)	0.85 (53%)
Standard deviation	0.00 (0%)	0.05 (12%)	0.18 (22%)	0.37 (23%)

Figure A.5: Load and store bandwidth for vertical image access pattern. The mean data for these cases are the fourth lines on the graphs in Figure 6.15. Layout is fixed at RSBCW. XORing is fixed at 0 levels. Sub-banks are fixed at 8. Lanes are varied.

Lanes	**1**	**2**	**4**	**8**
Peak Bandwidth	**0.2 GB/s**	**0.4 GB/s**	**0.8 GB/s**	**1.6 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.20 (100%)	0.22 (55%)	0.27 (33%)	0.27 (17%)
176 × 144	0.20 (100%)	0.40 (100%)	0.42 (52%)	0.52 (32%)
352 × 240	0.20 (100%)	0.30 (76%)	0.80 (100%)	0.86 (54%)
352 × 288	0.20 (100%)	0.30 (76%)	0.80 (100%)	0.86 (54%)
352 × 480	0.20 (100%)	0.30 (76%)	0.80 (100%)	0.86 (54%)
480 × 480	0.20 (100%)	0.38 (94%)	0.80 (100%)	1.60 (100%)
512 × 384	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.60 (100%)
544 × 480	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.29 (81%)
640 × 480	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.80 (50%)
704 × 480	0.20 (100%)	0.40 (100%)	0.79 (99%)	0.79 (50%)
720 × 400	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.09 (68%)
720 × 480	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.09 (68%)
800 × 600	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.88 (55%)
832 × 624	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.60 (100%)
1024 × 768	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.80 (50%)
1152 × 864	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.89 (56%)
1280 × 720	0.20 (100%)	0.40 (99%)	0.51 (64%)	0.62 (39%)
1280 × 1024	0.20 (100%)	0.40 (100%)	0.52 (64%)	0.63 (39%)
1600 × 1200	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.57 (98%)
1800 × 1440	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.49 (93%)
1920 × 1080	0.20 (100%)	0.40 (100%)	0.45 (57%)	0.58 (36%)
1920 × 1200	0.20 (100%)	0.40 (100%)	0.45 (57%)	0.58 (36%)
Median	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.86 (54%)
Mean	0.20 (100%)	0.38 (94%)	0.70 (88%)	0.97 (60%)
Standard deviation	0.00 (0%)	0.05 (12%)	0.17 (21%)	0.40 (25%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.20 (100%)	0.22 (55%)	0.27 (33%)	0.27 (17%)
176 × 144	0.20 (100%)	0.40 (100%)	0.42 (52%)	0.52 (32%)
352 × 240	0.20 (100%)	0.30 (76%)	0.80 (100%)	0.86 (54%)
352 × 288	0.20 (100%)	0.30 (76%)	0.80 (100%)	0.86 (54%)
352 × 480	0.20 (100%)	0.30 (76%)	0.80 (100%)	0.86 (54%)
480 × 480	0.20 (100%)	0.38 (94%)	0.80 (100%)	1.59 (99%)
512 × 384	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.60 (100%)
544 × 480	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.29 (81%)
640 × 480	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.80 (50%)
704 × 480	0.20 (100%)	0.40 (99%)	0.76 (95%)	0.76 (48%)
720 × 400	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.09 (68%)
720 × 480	0.20 (100%)	0.40 (100%)	0.78 (98%)	1.04 (65%)
800 × 600	0.20 (100%)	0.40 (100%)	0.79 (98%)	0.86 (54%)
832 × 624	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.60 (100%)
1024 × 768	0.20 (100%)	0.40 (100%)	0.80 (100%)	0.80 (50%)
1152 × 864	0.20 (100%)	0.40 (99%)	0.78 (97%)	0.84 (53%)
1280 × 720	0.20 (100%)	0.39 (98%)	0.50 (63%)	0.61 (38%)
1280 × 1024	0.20 (100%)	0.40 (100%)	0.52 (64%)	0.63 (39%)
1600 × 1200	0.20 (100%)	0.40 (100%)	0.79 (98%)	1.52 (95%)
1800 × 1440	0.20 (100%)	0.40 (100%)	0.80 (100%)	1.45 (91%)
1920 × 1080	0.20 (100%)	0.40 (100%)	0.45 (56%)	0.57 (36%)
1920 × 1200	0.20 (100%)	0.40 (100%)	0.45 (57%)	0.58 (36%)
Median	0.20 (100%)	0.40 (100%)	0.79 (99%)	0.86 (54%)
Mean	0.20 (100%)	0.38 (94%)	0.70 (87%)	0.95 (60%)
Standard deviation	0.00 (0%)	0.05 (12%)	0.17 (21%)	0.39 (25%)

Figure A.6: Load and store bandwidth for vertical image access pattern. The mean data for these cases are the fifth lines on the graphs in Figure 6.15. Layout is fixed at RSBCW. XORing is fixed at 0 levels. Sub-banks are fixed at 16. Lanes are varied.

Address generators **Peak Bandwidth**	**4** **0.8 GB/s**	**8** **1.6 GB/s**	**16** **3.2 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.26 (33%)	0.26 (16%)	0.26 (8%)
176 × 144	0.41 (51%)	0.49 (31%)	0.55 (17%)
352 × 240	0.60 (75%)	0.60 (37%)	0.61 (19%)
352 × 288	0.60 (75%)	0.60 (38%)	0.61 (19%)
352 × 480	0.60 (75%)	0.60 (38%)	0.61 (19%)
480 × 480	0.53 (66%)	0.63 (39%)	0.64 (20%)
512 × 384	0.40 (50%)	0.40 (25%)	0.40 (12%)
544 × 480	0.53 (66%)	0.63 (39%)	0.64 (20%)
640 × 480	0.32 (40%)	0.32 (20%)	0.33 (10%)
704 × 480	0.31 (38%)	0.31 (20%)	0.32 (10%)
720 × 400	0.55 (69%)	0.59 (37%)	0.60 (19%)
720 × 480	0.55 (69%)	0.58 (36%)	0.60 (19%)
800 × 600	0.48 (60%)	0.50 (31%)	0.51 (16%)
832 × 624	0.25 (31%)	0.26 (16%)	0.26 (8%)
1024 × 768	0.20 (25%)	0.20 (12%)	0.20 (6%)
1152 × 864	0.34 (42%)	0.34 (21%)	0.36 (11%)
1280 × 720	0.18 (22%)	0.19 (12%)	0.19 (6%)
1280 × 1024	0.18 (22%)	0.19 (12%)	0.19 (6%)
1600 × 1200	0.32 (40%)	0.33 (20%)	0.36 (11%)
1800 × 1440	0.39 (48%)	0.41 (25%)	0.41 (13%)
1920 × 1080	0.17 (21%)	0.18 (11%)	0.19 (6%)
1920 × 1200	0.17 (21%)	0.18 (11%)	0.19 (6%)
Median	0.36 (45%)	0.37 (23%)	0.38 (12%)
Mean	0.38 (47%)	0.40 (25%)	0.41 (13%)
Standard deviation	0.16 (19%)	0.17 (11%)	0.17 (5%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.24 (30%)	0.24 (15%)	0.24 (8%)
176 × 144	0.36 (46%)	0.40 (25%)	0.41 (13%)
352 × 240	0.27 (34%)	0.27 (17%)	0.28 (9%)
352 × 288	0.27 (34%)	0.27 (17%)	0.28 (9%)
352 × 480	0.27 (34%)	0.28 (17%)	0.28 (9%)
480 × 480	0.28 (36%)	0.31 (20%)	0.32 (10%)
512 × 384	0.18 (22%)	0.18 (11%)	0.18 (6%)
544 × 480	0.28 (36%)	0.31 (20%)	0.32 (10%)
640 × 480	0.15 (18%)	0.15 (9%)	0.15 (5%)
704 × 480	0.14 (18%)	0.14 (9%)	0.14 (4%)
720 × 400	0.26 (33%)	0.27 (17%)	0.27 (9%)
720 × 480	0.26 (33%)	0.27 (17%)	0.27 (9%)
800 × 600	0.23 (28%)	0.23 (14%)	0.23 (7%)
832 × 624	0.11 (14%)	0.12 (7%)	0.12 (4%)
1024 × 768	0.09 (11%)	0.09 (6%)	0.09 (3%)
1152 × 864	0.16 (20%)	0.16 (10%)	0.17 (5%)
1280 × 720	0.08 (10%)	0.09 (5%)	0.09 (3%)
1280 × 1024	0.08 (10%)	0.09 (5%)	0.09 (3%)
1600 × 1200	0.16 (20%)	0.16 (10%)	0.17 (5%)
1800 × 1440	0.18 (23%)	0.18 (12%)	0.19 (6%)
1920 × 1080	0.08 (10%)	0.09 (5%)	0.09 (3%)
1920 × 1200	0.08 (10%)	0.09 (5%)	0.09 (3%)
Median	0.18 (22%)	0.18 (11%)	0.18 (6%)
Mean	0.19 (24%)	0.20 (12%)	0.20 (6%)
Standard deviation	0.09 (11%)	0.09 (6%)	0.09 (3%)

Figure A.7: Load and store bandwidth for vertical image access pattern. The mean data for these cases are the first lines on the graphs in Figure 6.16. Layout is fixed at RSBCW. XORing is fixed at 0 levels. Sub-banks are fixed at 1. Address generation resources are varied.

Address generators **Peak Bandwidth**	**4** **0.8 GB/s**	**8** **1.6 GB/s**	**16** **3.2 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.27 (33%)	0.27 (17%)	0.27 (8%)
176 × 144	0.41 (51%)	0.49 (31%)	0.55 (17%)
352 × 240	0.77 (96%)	1.06 (66%)	1.01 (31%)
352 × 288	0.78 (97%)	1.08 (67%)	1.03 (32%)
352 × 480	0.79 (99%)	1.10 (69%)	1.04 (33%)
480 × 480	0.78 (98%)	1.06 (66%)	1.06 (33%)
512 × 384	0.80 (100%)	0.80 (50%)	0.80 (25%)
544 × 480	0.80 (100%)	1.26 (79%)	1.25 (39%)
640 × 480	0.61 (77%)	0.56 (35%)	0.61 (19%)
704 × 480	0.57 (72%)	0.53 (33%)	0.56 (17%)
720 × 400	0.56 (70%)	0.59 (37%)	0.60 (19%)
720 × 480	0.56 (70%)	0.59 (37%)	0.61 (19%)
800 × 600	0.48 (60%)	0.50 (31%)	0.51 (16%)
832 × 624	0.48 (60%)	0.49 (30%)	0.49 (15%)
1024 × 768	0.40 (50%)	0.40 (25%)	0.40 (12%)
1152 × 864	0.34 (42%)	0.34 (21%)	0.36 (11%)
1280 × 720	0.47 (59%)	0.56 (35%)	0.56 (18%)
1280 × 1024	0.48 (60%)	0.57 (36%)	0.57 (18%)
1600 × 1200	0.32 (40%)	0.33 (20%)	0.36 (11%)
1800 × 1440	0.47 (59%)	0.48 (30%)	0.48 (15%)
1920 × 1080	0.44 (55%)	0.55 (34%)	0.58 (18%)
1920 × 1200	0.44 (55%)	0.55 (34%)	0.58 (18%)
Median	0.48 (60%)	0.55 (35%)	0.57 (18%)
Mean	0.55 (68%)	0.64 (40%)	0.65 (20%)
Standard deviation	0.17 (21%)	0.28 (18%)	0.26 (8%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.27 (33%)	0.27 (17%)	0.27 (8%)
176 × 144	0.38 (47%)	0.45 (28%)	0.49 (15%)
352 × 240	0.48 (61%)	0.50 (31%)	0.49 (15%)
352 × 288	0.49 (61%)	0.50 (31%)	0.49 (15%)
352 × 480	0.50 (63%)	0.51 (32%)	0.50 (16%)
480 × 480	0.48 (60%)	0.56 (35%)	0.56 (18%)
512 × 384	0.36 (44%)	0.36 (22%)	0.36 (11%)
544 × 480	0.63 (78%)	0.63 (40%)	0.63 (20%)
640 × 480	0.29 (36%)	0.29 (18%)	0.29 (9%)
704 × 480	0.26 (33%)	0.25 (16%)	0.26 (8%)
720 × 400	0.27 (33%)	0.27 (17%)	0.28 (9%)
720 × 480	0.27 (33%)	0.27 (17%)	0.28 (9%)
800 × 600	0.23 (28%)	0.23 (14%)	0.23 (7%)
832 × 624	0.23 (28%)	0.23 (14%)	0.23 (7%)
1024 × 768	0.18 (22%)	0.18 (11%)	0.18 (6%)
1152 × 864	0.16 (20%)	0.16 (10%)	0.17 (5%)
1280 × 720	0.29 (36%)	0.29 (18%)	0.29 (9%)
1280 × 1024	0.29 (37%)	0.29 (18%)	0.29 (9%)
1600 × 1200	0.16 (20%)	0.16 (10%)	0.17 (5%)
1800 × 1440	0.22 (28%)	0.23 (14%)	0.23 (7%)
1920 × 1080	0.30 (38%)	0.30 (19%)	0.30 (9%)
1920 × 1200	0.30 (38%)	0.30 (19%)	0.30 (9%)
Median	0.29 (36%)	0.29 (18%)	0.29 (9%)
Mean	0.32 (40%)	0.33 (21%)	0.33 (10%)
Standard deviation	0.12 (16%)	0.14 (8%)	0.13 (4%)

Figure A.8: Load and store bandwidth for vertical image access pattern. The mean data for these cases are the second lines on the graphs in Figure 6.16. Layout is fixed at RSBCW. XORing is fixed at 0 levels. Sub-banks are fixed at 2. Address generation resources are varied.

Address generators	**4**	**8**	**16**
Peak Bandwidth	**0.8 GB/s**	**1.6 GB/s**	**3.2 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.27 (33%)	0.27 (17%)	0.27 (8%)
176 × 144	0.42 (52%)	0.51 (32%)	0.57 (18%)
352 × 240	0.77 (96%)	1.46 (91%)	1.46 (45%)
352 × 288	0.78 (97%)	1.48 (92%)	1.48 (46%)
352 × 480	0.79 (99%)	1.53 (95%)	1.53 (48%)
480 × 480	0.78 (98%)	1.53 (96%)	1.54 (48%)
512 × 384	0.80 (100%)	1.60 (100%)	1.60 (50%)
544 × 480	0.80 (100%)	1.28 (80%)	1.26 (39%)
640 × 480	0.80 (100%)	0.80 (50%)	1.05 (33%)
704 × 480	0.79 (99%)	0.79 (49%)	1.04 (33%)
720 × 400	0.79 (99%)	1.11 (69%)	1.04 (33%)
720 × 480	0.80 (100%)	1.14 (71%)	1.07 (33%)
800 × 600	0.79 (98%)	0.99 (62%)	1.01 (32%)
832 × 624	0.80 (99%)	0.79 (50%)	0.90 (28%)
1024 × 768	0.80 (100%)	0.80 (50%)	0.80 (25%)
1152 × 864	0.67 (83%)	0.60 (37%)	0.67 (21%)
1280 × 720	0.47 (59%)	0.56 (35%)	0.56 (18%)
1280 × 1024	0.48 (60%)	0.57 (36%)	0.57 (18%)
1600 × 1200	0.55 (69%)	0.59 (37%)	0.65 (20%)
1800 × 1440	0.47 (59%)	0.48 (30%)	0.48 (15%)
1920 × 1080	0.45 (56%)	0.57 (36%)	0.67 (21%)
1920 × 1200	0.45 (57%)	0.58 (36%)	0.67 (21%)
Median	0.78 (98%)	0.80 (50%)	0.95 (30%)
Mean	0.66 (82%)	0.91 (57%)	0.95 (30%)
Standard deviation	0.17 (22%)	0.41 (26%)	0.39 (12%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.27 (33%)	0.27 (17%)	0.27 (8%)
176 × 144	0.42 (52%)	0.51 (32%)	0.57 (18%)
352 × 240	0.71 (89%)	0.86 (54%)	0.86 (27%)
352 × 288	0.73 (91%)	0.88 (55%)	0.89 (28%)
352 × 480	0.76 (95%)	0.92 (57%)	0.92 (29%)
480 × 480	0.69 (86%)	0.72 (45%)	0.72 (23%)
512 × 384	0.71 (89%)	0.71 (44%)	0.71 (22%)
544 × 480	0.64 (80%)	0.64 (40%)	0.64 (20%)
640 × 480	0.55 (68%)	0.55 (34%)	0.56 (17%)
704 × 480	0.47 (59%)	0.48 (30%)	0.50 (16%)
720 × 400	0.51 (63%)	0.53 (33%)	0.53 (16%)
720 × 480	0.52 (65%)	0.54 (34%)	0.54 (17%)
800 × 600	0.47 (59%)	0.49 (30%)	0.49 (15%)
832 × 624	0.45 (57%)	0.45 (28%)	0.44 (14%)
1024 × 768	0.36 (44%)	0.36 (22%)	0.36 (11%)
1152 × 864	0.33 (41%)	0.31 (19%)	0.33 (10%)
1280 × 720	0.29 (36%)	0.29 (18%)	0.29 (9%)
1280 × 1024	0.29 (37%)	0.29 (18%)	0.29 (9%)
1600 × 1200	0.29 (37%)	0.30 (19%)	0.32 (10%)
1800 × 1440	0.22 (28%)	0.23 (14%)	0.23 (7%)
1920 × 1080	0.36 (45%)	0.37 (23%)	0.37 (12%)
1920 × 1200	0.36 (46%)	0.37 (23%)	0.38 (12%)
Median	0.46 (58%)	0.48 (30%)	0.49 (15%)
Mean	0.47 (59%)	0.50 (31%)	0.51 (16%)
Standard deviation	0.17 (21%)	0.21 (13%)	0.21 (7%)

Figure A.9: Load and store bandwidth for vertical image access pattern. The mean data for these cases are the third lines on the graphs in Figure 6.16. Layout is fixed at RSBCW. XORing is fixed at 0 levels. Sub-banks are fixed at 4. Address generation resources are varied.

Address generators Peak Bandwidth	4 0.8 GB/s	8 1.6 GB/s	16 3.2 GB/s
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.27 (33%)	0.27 (17%)	0.27 (8%)
176 × 144	0.42 (52%)	0.51 (32%)	0.57 (18%)
352 × 240	0.80 (100%)	1.60 (100%)	1.60 (50%)
352 × 288	0.78 (97%)	1.48 (92%)	1.48 (46%)
352 × 480	0.79 (99%)	1.53 (95%)	1.53 (48%)
480 × 480	0.78 (98%)	1.53 (96%)	1.58 (49%)
512 × 384	0.80 (100%)	1.60 (100%)	1.60 (50%)
544 × 480	0.80 (100%)	1.60 (100%)	1.64 (51%)
640 × 480	0.80 (100%)	0.80 (50%)	1.05 (33%)
704 × 480	0.79 (99%)	0.79 (49%)	1.04 (33%)
720 × 400	0.80 (100%)	1.60 (100%)	1.64 (51%)
720 × 480	0.80 (100%)	1.57 (98%)	1.59 (50%)
800 × 600	0.80 (100%)	1.58 (99%)	1.58 (49%)
832 × 624	0.80 (100%)	0.80 (50%)	0.90 (28%)
1024 × 768	0.80 (100%)	0.80 (50%)	0.80 (25%)
1152 × 864	0.80 (100%)	0.89 (56%)	1.14 (35%)
1280 × 720	0.51 (64%)	0.62 (39%)	0.69 (22%)
1280 × 1024	0.52 (64%)	0.63 (39%)	0.70 (22%)
1600 × 1200	0.80 (100%)	0.85 (53%)	1.10 (34%)
1800 × 1440	0.80 (100%)	0.92 (58%)	1.02 (32%)
1920 × 1080	0.45 (56%)	0.57 (36%)	0.67 (21%)
1920 × 1200	0.45 (57%)	0.58 (36%)	0.67 (21%)
Median	0.80 (100%)	0.87 (54%)	1.08 (34%)
Mean	0.70 (87%)	1.05 (66%)	1.13 (35%)
Standard deviation	0.17 (21%)	0.46 (29%)	0.43 (13%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.27 (33%)	0.27 (17%)	0.27 (8%)
176 × 144	0.42 (52%)	0.51 (32%)	0.57 (18%)
352 × 240	0.80 (100%)	1.46 (91%)	1.46 (45%)
352 × 288	0.73 (91%)	1.31 (82%)	1.31 (41%)
352 × 480	0.76 (95%)	1.41 (88%)	1.42 (44%)
480 × 480	0.75 (94%)	1.31 (82%)	1.32 (41%)
512 × 384	0.80 (100%)	1.42 (89%)	1.42 (44%)
544 × 480	0.80 (100%)	1.28 (80%)	1.25 (39%)
640 × 480	0.77 (96%)	0.77 (48%)	1.00 (31%)
704 × 480	0.76 (95%)	0.76 (48%)	0.91 (28%)
720 × 400	0.80 (100%)	1.02 (63%)	1.00 (31%)
720 × 480	0.78 (98%)	1.00 (63%)	0.99 (31%)
800 × 600	0.79 (98%)	1.48 (93%)	1.48 (46%)
832 × 624	0.80 (99%)	0.80 (50%)	0.81 (25%)
1024 × 768	0.71 (89%)	0.71 (44%)	0.71 (22%)
1152 × 864	0.78 (97%)	0.84 (53%)	1.06 (33%)
1280 × 720	0.50 (63%)	0.61 (38%)	0.68 (21%)
1280 × 1024	0.52 (64%)	0.63 (39%)	0.70 (22%)
1600 × 1200	0.79 (98%)	0.79 (50%)	0.85 (27%)
1800 × 1440	0.51 (64%)	0.53 (33%)	0.56 (18%)
1920 × 1080	0.36 (45%)	0.37 (23%)	0.37 (12%)
1920 × 1200	0.37 (46%)	0.37 (23%)	0.38 (12%)
Median	0.76 (95%)	0.79 (50%)	0.95 (30%)
Mean	0.66 (83%)	0.89 (56%)	0.93 (29%)
Standard deviation	0.18 (22%)	0.39 (24%)	0.38 (12%)

Figure A.10: Load and store bandwidth for vertical image access pattern. The mean data for these cases are the fourth lines on the graphs in Figure 6.16. Layout is fixed at RSBCW. XORing is fixed at 0 levels. Sub-banks are fixed at 8. Address generation resources are varied.

Address generators	**4**	**8**	**16**
Peak Bandwidth	**0.8 GB/s**	**1.6 GB/s**	**3.2 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.27 (33%)	0.27 (17%)	0.27 (8%)
176 × 144	0.42 (52%)	0.51 (32%)	0.57 (18%)
352 × 240	0.80 (100%)	1.60 (100%)	1.60 (50%)
352 × 288	0.80 (100%)	1.60 (100%)	1.60 (50%)
352 × 480	0.80 (100%)	1.60 (100%)	1.60 (50%)
480 × 480	0.80 (100%)	1.60 (100%)	1.66 (52%)
512 × 384	0.80 (100%)	1.60 (100%)	1.60 (50%)
544 × 480	0.80 (100%)	1.60 (100%)	1.65 (52%)
640 × 480	0.80 (100%)	0.80 (50%)	1.07 (33%)
704 × 480	0.79 (99%)	0.79 (49%)	1.04 (33%)
720 × 400	0.80 (100%)	1.60 (100%)	1.67 (52%)
720 × 480	0.80 (100%)	1.57 (98%)	1.63 (51%)
800 × 600	0.80 (100%)	1.58 (99%)	1.58 (49%)
832 × 624	0.80 (100%)	0.80 (50%)	0.90 (28%)
1024 × 768	0.80 (100%)	0.80 (50%)	0.80 (25%)
1152 × 864	0.80 (100%)	0.89 (56%)	1.14 (35%)
1280 × 720	0.51 (64%)	0.62 (39%)	0.69 (22%)
1280 × 1024	0.52 (64%)	0.63 (39%)	0.70 (22%)
1600 × 1200	0.80 (100%)	0.85 (53%)	1.10 (34%)
1800 × 1440	0.80 (100%)	1.25 (78%)	1.46 (46%)
1920 × 1080	0.45 (57%)	0.58 (36%)	0.67 (21%)
1920 × 1200	0.45 (57%)	0.58 (36%)	0.67 (21%)
Median	0.80 (100%)	0.87 (54%)	1.12 (35%)
Mean	0.70 (88%)	1.08 (67%)	1.17 (36%)
Standard deviation	0.17 (21%)	0.47 (30%)	0.45 (14%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.27 (33%)	0.27 (17%)	0.27 (8%)
176 × 144	0.42 (52%)	0.51 (32%)	0.57 (18%)
352 × 240	0.80 (100%)	1.46 (91%)	1.46 (45%)
352 × 288	0.80 (100%)	1.60 (100%)	1.60 (50%)
352 × 480	0.80 (100%)	1.60 (100%)	1.60 (50%)
480 × 480	0.80 (100%)	1.57 (98%)	1.62 (51%)
512 × 384	0.80 (100%)	1.60 (100%)	1.60 (50%)
544 × 480	0.80 (100%)	1.60 (100%)	1.64 (51%)
640 × 480	0.80 (100%)	0.80 (50%)	1.07 (33%)
704 × 480	0.76 (95%)	0.76 (48%)	0.99 (31%)
720 × 400	0.80 (100%)	1.46 (91%)	1.50 (47%)
720 × 480	0.78 (98%)	1.46 (91%)	1.49 (47%)
800 × 600	0.79 (98%)	1.48 (93%)	1.48 (46%)
832 × 624	0.80 (100%)	0.80 (50%)	0.90 (28%)
1024 × 768	0.80 (100%)	0.80 (50%)	0.80 (25%)
1152 × 864	0.78 (97%)	0.84 (53%)	1.06 (33%)
1280 × 720	0.50 (63%)	0.61 (38%)	0.68 (21%)
1280 × 1024	0.52 (64%)	0.63 (39%)	0.70 (22%)
1600 × 1200	0.79 (98%)	0.79 (50%)	0.85 (27%)
1800 × 1440	0.80 (100%)	1.25 (78%)	1.46 (46%)
1920 × 1080	0.45 (56%)	0.57 (36%)	0.67 (21%)
1920 × 1200	0.45 (57%)	0.58 (36%)	0.67 (21%)
Median	0.79 (99%)	0.82 (51%)	1.06 (33%)
Mean	0.70 (87%)	1.05 (65%)	1.12 (35%)
Standard deviation	0.17 (21%)	0.45 (28%)	0.43 (13%)

Figure A.11: Load and store bandwidth for vertical image access pattern. The mean data for these cases are the fifth lines on the graphs in Figure 6.16. Layout is fixed at RSBCW. XORing is fixed at 0 levels. Sub-banks are fixed at 16. Address generation resources are varied.

Lanes	**1**	**2**	**4**	**8**
Peak Bandwidth	**0.2 GB/s**	**0.4 GB/s**	**0.8 GB/s**	**1.6 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.14 (68%)	0.20 (50%)	0.25 (31%)	0.28 (18%)
176 × 144	0.13 (64%)	0.18 (45%)	0.22 (27%)	0.24 (15%)
352 × 240	0.12 (62%)	0.17 (42%)	0.20 (25%)	0.21 (13%)
352 × 288	0.12 (62%)	0.17 (42%)	0.19 (24%)	0.22 (14%)
352 × 480	0.12 (62%)	0.17 (42%)	0.20 (25%)	0.21 (13%)
480 × 480	0.12 (62%)	0.17 (42%)	0.20 (24%)	0.21 (13%)
512 × 384	0.12 (62%)	0.17 (42%)	0.20 (24%)	0.21 (13%)
544 × 480	0.12 (62%)	0.17 (42%)	0.19 (24%)	0.21 (13%)
640 × 480	0.12 (62%)	0.16 (40%)	0.19 (24%)	0.21 (13%)
704 × 480	0.12 (62%)	0.17 (42%)	0.19 (24%)	0.21 (13%)
720 × 400	0.12 (62%)	0.17 (41%)	0.19 (24%)	0.21 (13%)
720 × 480	0.12 (62%)	0.16 (41%)	0.19 (24%)	0.21 (13%)
800 × 600	0.12 (62%)	0.16 (41%)	0.19 (24%)	0.21 (13%)
832 × 624	0.12 (62%)	0.17 (41%)	0.19 (24%)	0.21 (13%)
1024 × 768	0.12 (62%)	0.17 (41%)	0.19 (24%)	0.21 (13%)
1152 × 864	0.12 (62%)	0.17 (41%)	0.19 (24%)	0.21 (13%)
1280 × 720	0.12 (62%)	0.16 (40%)	0.19 (24%)	0.21 (13%)
1280 × 1024	0.12 (62%)	0.16 (41%)	0.19 (24%)	0.21 (13%)
1600 × 1200	0.12 (62%)	0.16 (40%)	0.19 (23%)	0.21 (13%)
1800 × 1440	0.12 (62%)	0.17 (41%)	0.19 (24%)	0.21 (13%)
1920 × 1080	0.12 (62%)	0.16 (41%)	0.19 (24%)	0.21 (13%)
1920 × 1200	0.12 (62%)	0.17 (41%)	0.19 (24%)	0.21 (13%)
Median	0.12 (62%)	0.17 (41%)	0.19 (24%)	0.21 (13%)
Mean	0.12 (62%)	0.17 (42%)	0.20 (25%)	0.21 (13%)
Standard deviation	0.00 (1%)	0.01 (2%)	0.01 (2%)	0.02 (1%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.09 (43%)	0.10 (26%)	0.12 (14%)	0.12 (8%)
176 × 144	0.08 (40%)	0.09 (24%)	0.10 (13%)	0.11 (7%)
352 × 240	0.08 (39%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
352 × 288	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
352 × 480	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
480 × 480	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
512 × 384	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
544 × 480	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
640 × 480	0.07 (38%)	0.09 (21%)	0.09 (12%)	0.10 (6%)
704 × 480	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
720 × 400	0.08 (38%)	0.09 (22%)	0.09 (12%)	0.10 (6%)
720 × 480	0.08 (38%)	0.09 (22%)	0.09 (12%)	0.10 (6%)
800 × 600	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
832 × 624	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
1024 × 768	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
1152 × 864	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
1280 × 720	0.08 (38%)	0.09 (22%)	0.09 (12%)	0.10 (6%)
1280 × 1024	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
1600 × 1200	0.08 (38%)	0.09 (21%)	0.09 (12%)	0.10 (6%)
1800 × 1440	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
1920 × 1080	0.07 (38%)	0.09 (22%)	0.09 (12%)	0.10 (6%)
1920 × 1200	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
Median	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
Mean	0.08 (38%)	0.09 (22%)	0.10 (12%)	0.10 (6%)
Standard deviation	0.00 (1%)	0.00 (1%)	0.00 (1%)	0.01 (0%)

Figure A.12: Load and store bandwidth for randomized image access pattern. The mean data for these cases are the first lines on the graphs in Figure 6.21. Inner loop is unrolled twice and utilizes branch delay slots. Index array is aligned to the physical width in bits across all of the lanes. Sub-banks are fixed at 1. Lanes are varied.

Lanes Peak Bandwidth	1 0.2 GB/s	2 0.4 GB/s	4 0.8 GB/s	8 1.6 GB/s
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.15 (76%)	0.26 (64%)	0.36 (45%)	0.43 (27%)
176 × 144	0.15 (73%)	0.23 (59%)	0.31 (38%)	0.35 (22%)
352 × 240	0.14 (70%)	0.21 (53%)	0.26 (32%)	0.29 (18%)
352 × 288	0.14 (70%)	0.21 (52%)	0.25 (32%)	0.29 (18%)
352 × 480	0.14 (70%)	0.21 (52%)	0.26 (32%)	0.28 (18%)
480 × 480	0.14 (70%)	0.21 (52%)	0.25 (32%)	0.28 (17%)
512 × 384	0.14 (70%)	0.20 (51%)	0.25 (31%)	0.28 (17%)
544 × 480	0.14 (69%)	0.20 (51%)	0.25 (31%)	0.28 (17%)
640 × 480	0.14 (69%)	0.20 (50%)	0.25 (31%)	0.28 (17%)
704 × 480	0.14 (70%)	0.20 (51%)	0.25 (31%)	0.27 (17%)
720 × 400	0.14 (69%)	0.20 (51%)	0.24 (31%)	0.27 (17%)
720 × 480	0.14 (70%)	0.20 (51%)	0.24 (30%)	0.28 (18%)
800 × 600	0.14 (69%)	0.20 (51%)	0.25 (31%)	0.27 (17%)
832 × 624	0.14 (69%)	0.20 (51%)	0.25 (31%)	0.27 (17%)
1024 × 768	0.14 (70%)	0.20 (51%)	0.25 (31%)	0.27 (17%)
1152 × 864	0.14 (69%)	0.20 (51%)	0.25 (31%)	0.28 (17%)
1280 × 720	0.14 (69%)	0.20 (51%)	0.24 (31%)	0.27 (17%)
1280 × 1024	0.14 (70%)	0.20 (51%)	0.25 (31%)	0.27 (17%)
1600 × 1200	0.14 (69%)	0.20 (50%)	0.24 (30%)	0.27 (17%)
1800 × 1440	0.14 (69%)	0.20 (51%)	0.24 (30%)	0.27 (17%)
1920 × 1080	0.14 (68%)	0.20 (50%)	0.24 (30%)	0.26 (16%)
1920 × 1200	0.14 (69%)	0.20 (51%)	0.25 (31%)	0.27 (17%)
Median	0.14 (69%)	0.20 (51%)	0.25 (31%)	0.27 (17%)
Mean	0.14 (70%)	0.21 (52%)	0.26 (32%)	0.29 (18%)
Standard deviation	0.00 (2%)	0.01 (3%)	0.03 (3%)	0.04 (2%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.12 (59%)	0.15 (38%)	0.18 (23%)	0.20 (12%)
176 × 144	0.10 (52%)	0.13 (33%)	0.15 (19%)	0.16 (10%)
352 × 240	0.10 (50%)	0.12 (30%)	0.13 (17%)	0.14 (9%)
352 × 288	0.10 (49%)	0.12 (30%)	0.13 (16%)	0.14 (9%)
352 × 480	0.10 (49%)	0.12 (30%)	0.13 (17%)	0.14 (9%)
480 × 480	0.10 (48%)	0.12 (30%)	0.13 (16%)	0.14 (9%)
512 × 384	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (9%)
544 × 480	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (9%)
640 × 480	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (8%)
704 × 480	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (9%)
720 × 400	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.13 (8%)
720 × 480	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (9%)
800 × 600	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (8%)
832 × 624	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (8%)
1024 × 768	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (8%)
1152 × 864	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (9%)
1280 × 720	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (9%)
1280 × 1024	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (8%)
1600 × 1200	0.10 (48%)	0.12 (29%)	0.12 (16%)	0.14 (8%)
1800 × 1440	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.13 (8%)
1920 × 1080	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.13 (8%)
1920 × 1200	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (8%)
Median	0.10 (48%)	0.12 (29%)	0.13 (16%)	0.14 (9%)
Mean	0.10 (49%)	0.12 (30%)	0.13 (17%)	0.14 (9%)
Standard deviation	0.00 (2%)	0.01 (2%)	0.01 (2%)	0.01 (1%)

Figure A.13: Load and store bandwidth for randomized image access pattern. The mean data for these cases are the second lines on the graphs in Figure 6.21. Inner loop is unrolled twice and utilizes branch delay slots. Index array is aligned to the physical width in bits across all of the lanes. Sub-banks are fixed at 2. Lanes are varied.

Lanes	**1**	**2**	**4**	**8**
Peak Bandwidth	**0.2 GB/s**	**0.4 GB/s**	**0.8 GB/s**	**1.6 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.16 (79%)	0.29 (74%)	0.48 (60%)	0.63 (40%)
176 × 144	0.15 (78%)	0.28 (69%)	0.41 (51%)	0.50 (31%)
352 × 240	0.15 (76%)	0.25 (63%)	0.34 (42%)	0.39 (24%)
352 × 288	0.15 (75%)	0.25 (62%)	0.32 (40%)	0.38 (24%)
352 × 480	0.15 (74%)	0.25 (62%)	0.32 (40%)	0.37 (23%)
480 × 480	0.15 (74%)	0.24 (60%)	0.32 (40%)	0.35 (22%)
512 × 384	0.15 (74%)	0.24 (60%)	0.31 (39%)	0.36 (22%)
544 × 480	0.15 (74%)	0.24 (60%)	0.32 (40%)	0.36 (22%)
640 × 480	0.15 (74%)	0.24 (59%)	0.31 (38%)	0.35 (22%)
704 × 480	0.15 (74%)	0.24 (60%)	0.31 (39%)	0.35 (22%)
720 × 400	0.15 (74%)	0.24 (60%)	0.31 (38%)	0.35 (22%)
720 × 480	0.15 (74%)	0.24 (60%)	0.31 (39%)	0.36 (22%)
800 × 600	0.15 (74%)	0.24 (59%)	0.31 (38%)	0.35 (22%)
832 × 624	0.15 (74%)	0.24 (59%)	0.31 (38%)	0.35 (22%)
1024 × 768	0.15 (74%)	0.24 (59%)	0.31 (38%)	0.35 (22%)
1152 × 864	0.15 (74%)	0.24 (59%)	0.31 (39%)	0.35 (22%)
1280 × 720	0.15 (74%)	0.24 (59%)	0.31 (38%)	0.35 (22%)
1280 × 1024	0.15 (74%)	0.24 (59%)	0.31 (39%)	0.35 (22%)
1600 × 1200	0.15 (74%)	0.23 (59%)	0.30 (37%)	0.35 (22%)
1800 × 1440	0.15 (74%)	0.24 (59%)	0.30 (38%)	0.34 (21%)
1920 × 1080	0.15 (73%)	0.24 (59%)	0.30 (38%)	0.34 (21%)
1920 × 1200	0.15 (74%)	0.23 (59%)	0.30 (38%)	0.34 (21%)
Median	0.15 (74%)	0.24 (60%)	0.31 (39%)	0.35 (22%)
Mean	0.15 (75%)	0.24 (61%)	0.32 (40%)	0.37 (23%)
Standard deviation	0.00 (1%)	0.01 (4%)	0.04 (5%)	0.07 (4%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.16 (78%)	0.29 (72%)	0.46 (57%)	0.59 (37%)
176 × 144	0.13 (66%)	0.19 (48%)	0.23 (29%)	0.25 (16%)
352 × 240	0.12 (60%)	0.16 (40%)	0.19 (23%)	0.20 (13%)
352 × 288	0.12 (60%)	0.16 (40%)	0.18 (22%)	0.20 (12%)
352 × 480	0.12 (59%)	0.16 (40%)	0.18 (22%)	0.20 (12%)
480 × 480	0.12 (58%)	0.15 (38%)	0.18 (22%)	0.19 (12%)
512 × 384	0.12 (58%)	0.15 (38%)	0.17 (22%)	0.19 (12%)
544 × 480	0.12 (58%)	0.15 (38%)	0.18 (22%)	0.19 (12%)
640 × 480	0.12 (58%)	0.15 (37%)	0.17 (22%)	0.19 (12%)
704 × 480	0.12 (58%)	0.15 (38%)	0.17 (22%)	0.19 (12%)
720 × 400	0.12 (58%)	0.15 (38%)	0.17 (22%)	0.19 (12%)
720 × 480	0.12 (59%)	0.15 (38%)	0.17 (22%)	0.19 (12%)
800 × 600	0.12 (58%)	0.15 (38%)	0.17 (22%)	0.18 (11%)
832 × 624	0.12 (58%)	0.15 (37%)	0.17 (22%)	0.18 (11%)
1024 × 768	0.12 (58%)	0.15 (38%)	0.17 (21%)	0.18 (11%)
1152 × 864	0.12 (58%)	0.15 (38%)	0.17 (22%)	0.19 (12%)
1280 × 720	0.12 (58%)	0.15 (38%)	0.17 (21%)	0.19 (12%)
1280 × 1024	0.12 (58%)	0.15 (38%)	0.17 (22%)	0.18 (11%)
1600 × 1200	0.12 (58%)	0.15 (38%)	0.17 (21%)	0.18 (12%)
1800 × 1440	0.12 (58%)	0.15 (38%)	0.17 (21%)	0.18 (11%)
1920 × 1080	0.12 (58%)	0.15 (38%)	0.17 (22%)	0.18 (11%)
1920 × 1200	0.12 (58%)	0.15 (37%)	0.17 (21%)	0.18 (11%)
Median	0.12 (58%)	0.15 (38%)	0.17 (22%)	0.19 (12%)
Mean	0.12 (60%)	0.16 (40%)	0.19 (24%)	0.21 (13%)
Standard deviation	0.01 (5%)	0.03 (7%)	0.06 (8%)	0.09 (5%)

Figure A.14: Load and store bandwidth for randomized image access pattern. The mean data for these cases are the third lines on the graphs in Figure 6.21. Inner loop is unrolled twice and utilizes branch delay slots. Index array is aligned to the physical width in bits across all of the lanes. Sub-banks are fixed at 4. Lanes are varied.

Lanes **Peak Bandwidth**	**1** **0.2 GB/s**	**2** **0.4 GB/s**	**4** **0.8 GB/s**	**8** **1.6 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.16 (79%)	0.30 (74%)	0.48 (60%)	0.63 (40%)
176 × 144	0.16 (79%)	0.30 (74%)	0.47 (59%)	0.62 (39%)
352 × 240	0.16 (78%)	0.28 (70%)	0.41 (52%)	0.50 (31%)
352 × 288	0.16 (78%)	0.28 (70%)	0.40 (50%)	0.49 (31%)
352 × 480	0.15 (78%)	0.27 (68%)	0.39 (49%)	0.47 (29%)
480 × 480	0.15 (77%)	0.27 (67%)	0.38 (48%)	0.44 (27%)
512 × 384	0.15 (77%)	0.27 (67%)	0.38 (47%)	0.45 (28%)
544 × 480	0.15 (77%)	0.27 (67%)	0.38 (48%)	0.45 (28%)
640 × 480	0.15 (76%)	0.27 (66%)	0.37 (46%)	0.44 (27%)
704 × 480	0.15 (77%)	0.27 (67%)	0.37 (47%)	0.44 (27%)
720 × 400	0.15 (77%)	0.27 (67%)	0.37 (47%)	0.43 (27%)
720 × 480	0.15 (77%)	0.27 (66%)	0.37 (46%)	0.44 (27%)
800 × 600	0.15 (76%)	0.26 (66%)	0.37 (46%)	0.43 (27%)
832 × 624	0.15 (77%)	0.27 (66%)	0.37 (46%)	0.43 (27%)
1024 × 768	0.15 (77%)	0.26 (66%)	0.36 (45%)	0.42 (26%)
1152 × 864	0.15 (76%)	0.26 (66%)	0.36 (46%)	0.43 (27%)
1280 × 720	0.15 (76%)	0.26 (65%)	0.36 (45%)	0.43 (27%)
1280 × 1024	0.15 (76%)	0.26 (66%)	0.37 (46%)	0.42 (26%)
1600 × 1200	0.15 (76%)	0.26 (65%)	0.35 (44%)	0.42 (26%)
1800 × 1440	0.15 (76%)	0.26 (66%)	0.36 (45%)	0.41 (26%)
1920 × 1080	0.15 (76%)	0.26 (66%)	0.36 (45%)	0.41 (26%)
1920 × 1200	0.15 (76%)	0.26 (65%)	0.36 (44%)	0.41 (26%)
Median	0.15 (77%)	0.27 (66%)	0.37 (46%)	0.43 (27%)
Mean	0.15 (77%)	0.27 (67%)	0.38 (48%)	0.46 (28%)
Standard deviation	0.00 (1%)	0.01 (3%)	0.03 (4%)	0.06 (4%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.16 (79%)	0.29 (73%)	0.47 (59%)	0.62 (39%)
176 × 144	0.16 (79%)	0.29 (73%)	0.46 (58%)	0.59 (37%)
352 × 240	0.14 (70%)	0.21 (53%)	0.26 (33%)	0.29 (18%)
352 × 288	0.14 (70%)	0.20 (51%)	0.25 (31%)	0.28 (17%)
352 × 480	0.14 (68%)	0.20 (50%)	0.24 (30%)	0.27 (17%)
480 × 480	0.13 (66%)	0.19 (48%)	0.24 (30%)	0.25 (16%)
512 × 384	0.13 (67%)	0.19 (48%)	0.23 (29%)	0.25 (16%)
544 × 480	0.14 (68%)	0.20 (49%)	0.24 (30%)	0.25 (16%)
640 × 480	0.13 (66%)	0.19 (48%)	0.23 (29%)	0.25 (16%)
704 × 480	0.14 (68%)	0.20 (49%)	0.23 (29%)	0.25 (16%)
720 × 400	0.13 (67%)	0.19 (48%)	0.23 (29%)	0.25 (16%)
720 × 480	0.14 (68%)	0.19 (48%)	0.23 (29%)	0.25 (16%)
800 × 600	0.13 (66%)	0.19 (48%)	0.23 (29%)	0.25 (16%)
832 × 624	0.13 (67%)	0.19 (48%)	0.23 (29%)	0.25 (16%)
1024 × 768	0.13 (66%)	0.19 (47%)	0.22 (28%)	0.24 (15%)
1152 × 864	0.13 (66%)	0.19 (48%)	0.23 (28%)	0.25 (16%)
1280 × 720	0.13 (66%)	0.19 (47%)	0.22 (28%)	0.25 (16%)
1280 × 1024	0.13 (66%)	0.19 (47%)	0.23 (28%)	0.24 (15%)
1600 × 1200	0.13 (66%)	0.19 (47%)	0.22 (27%)	0.25 (15%)
1800 × 1440	0.13 (66%)	0.19 (47%)	0.22 (27%)	0.24 (15%)
1920 × 1080	0.13 (66%)	0.19 (47%)	0.22 (28%)	0.24 (15%)
1920 × 1200	0.13 (66%)	0.18 (46%)	0.22 (27%)	0.24 (15%)
Median	0.13 (67%)	0.19 (48%)	0.23 (29%)	0.25 (16%)
Mean	0.14 (68%)	0.20 (50%)	0.25 (32%)	0.28 (18%)
Standard deviation	0.01 (4%)	0.03 (8%)	0.07 (9%)	0.10 (7%)

Figure A.15: Load and store bandwidth for randomized image access pattern. The mean data for these cases are the fourth lines on the graphs in Figure 6.21. Inner loop is unrolled twice and utilizes branch delay slots. Index array is aligned to the physical width in bits across all of the lanes. Sub-banks are fixed at 8. Lanes are varied.

Lanes Peak Bandwidth	1 0.2 GB/s	2 0.4 GB/s	4 0.8 GB/s	8 1.6 GB/s
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.16 (79%)	0.30 (74%)	0.48 (60%)	0.64 (40%)
176 × 144	0.16 (80%)	0.30 (74%)	0.48 (59%)	0.62 (39%)
352 × 240	0.16 (79%)	0.29 (73%)	0.46 (57%)	0.58 (36%)
352 × 288	0.16 (79%)	0.29 (73%)	0.45 (56%)	0.58 (36%)
352 × 480	0.16 (78%)	0.29 (73%)	0.45 (56%)	0.56 (35%)
480 × 480	0.16 (78%)	0.28 (71%)	0.44 (55%)	0.53 (33%)
512 × 384	0.16 (78%)	0.28 (71%)	0.44 (54%)	0.54 (34%)
544 × 480	0.16 (78%)	0.28 (71%)	0.43 (54%)	0.53 (33%)
640 × 480	0.16 (78%)	0.28 (70%)	0.42 (53%)	0.51 (32%)
704 × 480	0.16 (78%)	0.28 (71%)	0.43 (53%)	0.52 (33%)
720 × 400	0.16 (78%)	0.28 (71%)	0.42 (53%)	0.51 (32%)
720 × 480	0.16 (78%)	0.28 (71%)	0.42 (53%)	0.52 (32%)
800 × 600	0.16 (78%)	0.28 (70%)	0.42 (52%)	0.51 (32%)
832 × 624	0.16 (78%)	0.28 (70%)	0.42 (52%)	0.51 (32%)
1024 × 768	0.16 (78%)	0.28 (70%)	0.41 (51%)	0.49 (31%)
1152 × 864	0.16 (78%)	0.28 (70%)	0.41 (52%)	0.50 (31%)
1280 × 720	0.16 (78%)	0.28 (70%)	0.41 (51%)	0.50 (31%)
1280 × 1024	0.16 (78%)	0.28 (70%)	0.41 (52%)	0.49 (31%)
1600 × 1200	0.16 (78%)	0.28 (69%)	0.40 (50%)	0.49 (31%)
1800 × 1440	0.16 (78%)	0.28 (70%)	0.40 (50%)	0.48 (30%)
1920 × 1080	0.16 (78%)	0.28 (70%)	0.41 (51%)	0.49 (30%)
1920 × 1200	0.16 (78%)	0.28 (69%)	0.40 (50%)	0.48 (30%)
Median	0.16 (78%)	0.28 (70%)	0.42 (53%)	0.51 (32%)
Mean	0.16 (78%)	0.28 (71%)	0.43 (53%)	0.52 (33%)
Standard deviation	0.00 (0%)	0.01 (2%)	0.02 (3%)	0.04 (3%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**			
128 × 96	0.16 (79%)	0.30 (74%)	0.48 (60%)	0.64 (40%)
176 × 144	0.16 (80%)	0.30 (74%)	0.48 (59%)	0.62 (39%)
352 × 240	0.15 (76%)	0.26 (65%)	0.36 (45%)	0.42 (26%)
352 × 288	0.15 (76%)	0.25 (63%)	0.34 (43%)	0.41 (26%)
352 × 480	0.15 (75%)	0.25 (62%)	0.32 (40%)	0.37 (23%)
480 × 480	0.15 (73%)	0.23 (59%)	0.31 (38%)	0.34 (21%)
512 × 384	0.15 (73%)	0.24 (59%)	0.31 (39%)	0.35 (22%)
544 × 480	0.15 (73%)	0.24 (59%)	0.31 (39%)	0.34 (21%)
640 × 480	0.14 (72%)	0.23 (58%)	0.30 (38%)	0.33 (20%)
704 × 480	0.15 (73%)	0.24 (59%)	0.30 (38%)	0.34 (21%)
720 × 400	0.15 (73%)	0.23 (58%)	0.30 (38%)	0.33 (21%)
720 × 480	0.15 (73%)	0.23 (58%)	0.30 (37%)	0.33 (21%)
800 × 600	0.14 (72%)	0.23 (57%)	0.29 (36%)	0.32 (20%)
832 × 624	0.15 (73%)	0.23 (57%)	0.29 (37%)	0.33 (21%)
1024 × 768	0.14 (72%)	0.22 (56%)	0.28 (35%)	0.32 (20%)
1152 × 864	0.14 (72%)	0.23 (56%)	0.29 (36%)	0.32 (20%)
1280 × 720	0.14 (72%)	0.22 (56%)	0.28 (35%)	0.32 (20%)
1280 × 1024	0.14 (72%)	0.22 (55%)	0.29 (36%)	0.32 (20%)
1600 × 1200	0.14 (71%)	0.22 (56%)	0.28 (35%)	0.32 (20%)
1800 × 1440	0.14 (71%)	0.22 (55%)	0.28 (35%)	0.30 (19%)
1920 × 1080	0.14 (72%)	0.22 (56%)	0.28 (35%)	0.32 (20%)
1920 × 1200	0.14 (72%)	0.22 (55%)	0.28 (35%)	0.31 (19%)
Median	0.15 (73%)	0.23 (58%)	0.30 (37%)	0.33 (21%)
Mean	0.15 (74%)	0.24 (60%)	0.32 (39%)	0.36 (23%)
Standard deviation	0.00 (2%)	0.02 (5%)	0.06 (7%)	0.09 (6%)

Figure A.16: Load and store bandwidth for randomized image access pattern. The mean data for these cases are the fifth lines on the graphs in Figure 6.21. Inner loop is unrolled twice and utilizes branch delay slots. Index array is aligned to the physical width in bits across all of the lanes. Sub-banks are fixed at 16. Lanes are varied.

Address generators	**4**	**8**	**16**
Peak Bandwidth	**0.8 GB/s**	**1.6 GB/s**	**3.2 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.25 (31%)	0.26 (16%)	0.26 (8%)
176 × 144	0.22 (27%)	0.23 (14%)	0.23 (7%)
352 × 240	0.20 (25%)	0.21 (13%)	0.21 (6%)
352 × 288	0.19 (24%)	0.20 (13%)	0.20 (6%)
352 × 480	0.20 (25%)	0.21 (13%)	0.21 (6%)
480 × 480	0.20 (24%)	0.21 (13%)	0.21 (6%)
512 × 384	0.20 (24%)	0.21 (13%)	0.21 (6%)
544 × 480	0.19 (24%)	0.20 (13%)	0.20 (6%)
640 × 480	0.19 (24%)	0.20 (12%)	0.20 (6%)
704 × 480	0.19 (24%)	0.20 (13%)	0.20 (6%)
720 × 400	0.19 (24%)	0.20 (13%)	0.20 (6%)
720 × 480	0.19 (24%)	0.20 (12%)	0.20 (6%)
800 × 600	0.19 (24%)	0.20 (13%)	0.20 (6%)
832 × 624	0.19 (24%)	0.20 (13%)	0.20 (6%)
1024 × 768	0.19 (24%)	0.20 (13%)	0.20 (6%)
1152 × 864	0.19 (24%)	0.20 (13%)	0.20 (6%)
1280 × 720	0.19 (24%)	0.20 (12%)	0.20 (6%)
1280 × 1024	0.19 (24%)	0.20 (13%)	0.20 (6%)
1600 × 1200	0.19 (23%)	0.20 (12%)	0.20 (6%)
1800 × 1440	0.19 (24%)	0.20 (13%)	0.20 (6%)
1920 × 1080	0.19 (24%)	0.20 (12%)	0.20 (6%)
1920 × 1200	0.19 (24%)	0.20 (13%)	0.20 (6%)
Median	0.19 (24%)	0.20 (13%)	0.20 (6%)
Mean	0.20 (25%)	0.21 (13%)	0.21 (6%)
Standard deviation	0.01 (2%)	0.01 (1%)	0.01 (0%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.12 (14%)	0.12 (7%)	0.12 (4%)
176 × 144	0.10 (13%)	0.11 (7%)	0.11 (3%)
352 × 240	0.10 (12%)	0.10 (6%)	0.10 (3%)
352 × 288	0.10 (12%)	0.10 (6%)	0.10 (3%)
352 × 480	0.10 (12%)	0.10 (6%)	0.10 (3%)
480 × 480	0.10 (12%)	0.10 (6%)	0.10 (3%)
512 × 384	0.10 (12%)	0.10 (6%)	0.10 (3%)
544 × 480	0.10 (12%)	0.10 (6%)	0.10 (3%)
640 × 480	0.09 (12%)	0.10 (6%)	0.10 (3%)
704 × 480	0.10 (12%)	0.10 (6%)	0.10 (3%)
720 × 400	0.09 (12%)	0.10 (6%)	0.10 (3%)
720 × 480	0.09 (12%)	0.10 (6%)	0.10 (3%)
800 × 600	0.10 (12%)	0.10 (6%)	0.10 (3%)
832 × 624	0.10 (12%)	0.10 (6%)	0.10 (3%)
1024 × 768	0.10 (12%)	0.10 (6%)	0.10 (3%)
1152 × 864	0.10 (12%)	0.10 (6%)	0.10 (3%)
1280 × 720	0.09 (12%)	0.10 (6%)	0.10 (3%)
1280 × 1024	0.10 (12%)	0.10 (6%)	0.10 (3%)
1600 × 1200	0.09 (12%)	0.10 (6%)	0.10 (3%)
1800 × 1440	0.10 (12%)	0.10 (6%)	0.10 (3%)
1920 × 1080	0.09 (12%)	0.10 (6%)	0.10 (3%)
1920 × 1200	0.10 (12%)	0.10 (6%)	0.10 (3%)
Median	0.10 (12%)	0.10 (6%)	0.10 (3%)
Mean	0.10 (12%)	0.10 (6%)	0.10 (3%)
Standard deviation	0.00 (1%)	0.00 (0%)	0.00 (0%)

Figure A.17: Load and store bandwidth for randomized image access pattern. The mean data for these cases are the first lines on the graphs in Figure 6.22. Inner loop is unrolled twice and utilizes branch delay slots. Index array is 256-bit aligned. Sub-banks are fixed at 1. Lanes are fixed at 4. Address generation resources are varied.

Address generators	**4**	**8**	**16**
Peak Bandwidth	**0.8 GB/s**	**1.6 GB/s**	**3.2 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.36 (45%)	0.39 (24%)	0.39 (12%)
176 × 144	0.31 (38%)	0.33 (20%)	0.33 (10%)
352 × 240	0.26 (32%)	0.27 (17%)	0.27 (9%)
352 × 288	0.25 (32%)	0.27 (17%)	0.27 (8%)
352 × 480	0.26 (32%)	0.27 (17%)	0.27 (8%)
480 × 480	0.25 (32%)	0.27 (17%)	0.27 (8%)
512 × 384	0.25 (31%)	0.27 (17%)	0.27 (8%)
544 × 480	0.25 (31%)	0.27 (17%)	0.27 (8%)
640 × 480	0.25 (31%)	0.26 (16%)	0.26 (8%)
704 × 480	0.25 (31%)	0.26 (16%)	0.26 (8%)
720 × 400	0.24 (31%)	0.26 (16%)	0.26 (8%)
720 × 480	0.24 (30%)	0.26 (16%)	0.26 (8%)
800 × 600	0.25 (31%)	0.26 (16%)	0.26 (8%)
832 × 624	0.25 (31%)	0.26 (16%)	0.26 (8%)
1024 × 768	0.25 (31%)	0.26 (16%)	0.26 (8%)
1152 × 864	0.25 (31%)	0.26 (16%)	0.26 (8%)
1280 × 720	0.24 (31%)	0.26 (16%)	0.26 (8%)
1280 × 1024	0.25 (31%)	0.26 (16%)	0.26 (8%)
1600 × 1200	0.24 (30%)	0.25 (16%)	0.25 (8%)
1800 × 1440	0.24 (30%)	0.26 (16%)	0.26 (8%)
1920 × 1080	0.24 (30%)	0.26 (16%)	0.26 (8%)
1920 × 1200	0.25 (31%)	0.26 (16%)	0.26 (8%)
Median	0.25 (31%)	0.26 (16%)	0.26 (8%)
Mean	0.26 (32%)	0.27 (17%)	0.27 (8%)
Standard deviation	0.03 (3%)	0.03 (2%)	0.03 (1%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.18 (23%)	0.19 (12%)	0.19 (6%)
176 × 144	0.15 (19%)	0.16 (10%)	0.16 (5%)
352 × 240	0.13 (17%)	0.14 (9%)	0.14 (4%)
352 × 288	0.13 (16%)	0.14 (8%)	0.14 (4%)
352 × 480	0.13 (17%)	0.14 (9%)	0.14 (4%)
480 × 480	0.13 (16%)	0.14 (8%)	0.14 (4%)
512 × 384	0.13 (16%)	0.14 (8%)	0.14 (4%)
544 × 480	0.13 (16%)	0.14 (8%)	0.14 (4%)
640 × 480	0.13 (16%)	0.13 (8%)	0.13 (4%)
704 × 480	0.13 (16%)	0.13 (8%)	0.13 (4%)
720 × 400	0.13 (16%)	0.13 (8%)	0.13 (4%)
720 × 480	0.13 (16%)	0.13 (8%)	0.13 (4%)
800 × 600	0.13 (16%)	0.13 (8%)	0.13 (4%)
832 × 624	0.13 (16%)	0.13 (8%)	0.13 (4%)
1024 × 768	0.13 (16%)	0.13 (8%)	0.13 (4%)
1152 × 864	0.13 (16%)	0.13 (8%)	0.13 (4%)
1280 × 720	0.13 (16%)	0.13 (8%)	0.13 (4%)
1280 × 1024	0.13 (16%)	0.13 (8%)	0.13 (4%)
1600 × 1200	0.12 (16%)	0.13 (8%)	0.13 (4%)
1800 × 1440	0.13 (16%)	0.13 (8%)	0.13 (4%)
1920 × 1080	0.13 (16%)	0.13 (8%)	0.13 (4%)
1920 × 1200	0.13 (16%)	0.14 (8%)	0.14 (4%)
Median	0.13 (16%)	0.13 (8%)	0.13 (4%)
Mean	0.13 (17%)	0.14 (9%)	0.14 (4%)
Standard deviation	0.01 (2%)	0.01 (1%)	0.01 (0%)

Figure A.18: Load and store bandwidth for randomized image access pattern. The mean data for these cases are the second lines on the graphs in Figure 6.22. Inner loop is unrolled twice and utilizes branch delay slots. Index array is 256-bit aligned. Sub-banks are fixed at 2. Lanes are fixed at 4. Address generation resources are varied.

Address generators **Peak Bandwidth**	**4** **0.8 GB/s**	**8** **1.6 GB/s**	**16** **3.2 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.48 (60%)	0.56 (35%)	0.56 (17%)
176 × 144	0.41 (51%)	0.45 (28%)	0.45 (14%)
352 × 240	0.34 (42%)	0.36 (23%)	0.36 (11%)
352 × 288	0.32 (40%)	0.35 (22%)	0.35 (11%)
352 × 480	0.32 (40%)	0.34 (21%)	0.34 (11%)
480 × 480	0.32 (40%)	0.34 (21%)	0.34 (11%)
512 × 384	0.31 (39%)	0.33 (21%)	0.33 (10%)
544 × 480	0.32 (40%)	0.34 (21%)	0.34 (11%)
640 × 480	0.31 (38%)	0.33 (21%)	0.33 (10%)
704 × 480	0.31 (39%)	0.33 (21%)	0.33 (10%)
720 × 400	0.31 (38%)	0.33 (21%)	0.33 (10%)
720 × 480	0.31 (39%)	0.33 (21%)	0.33 (10%)
800 × 600	0.31 (38%)	0.33 (21%)	0.33 (10%)
832 × 624	0.31 (38%)	0.33 (21%)	0.33 (10%)
1024 × 768	0.31 (38%)	0.33 (20%)	0.33 (10%)
1152 × 864	0.31 (39%)	0.33 (21%)	0.33 (10%)
1280 × 720	0.31 (38%)	0.33 (20%)	0.33 (10%)
1280 × 1024	0.31 (39%)	0.33 (20%)	0.33 (10%)
1600 × 1200	0.30 (37%)	0.32 (20%)	0.32 (10%)
1800 × 1440	0.30 (38%)	0.32 (20%)	0.32 (10%)
1920 × 1080	0.30 (38%)	0.33 (20%)	0.33 (10%)
1920 × 1200	0.30 (38%)	0.32 (20%)	0.32 (10%)
Median	0.31 (39%)	0.33 (21%)	0.33 (10%)
Mean	0.32 (40%)	0.35 (22%)	0.35 (11%)
Standard deviation	0.04 (5%)	0.05 (3%)	0.05 (2%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.46 (57%)	0.52 (33%)	0.52 (16%)
176 × 144	0.23 (29%)	0.24 (15%)	0.24 (8%)
352 × 240	0.19 (23%)	0.19 (12%)	0.19 (6%)
352 × 288	0.18 (22%)	0.19 (12%)	0.19 (6%)
352 × 480	0.18 (22%)	0.19 (12%)	0.19 (6%)
480 × 480	0.18 (22%)	0.19 (12%)	0.19 (6%)
512 × 384	0.17 (22%)	0.18 (11%)	0.18 (6%)
544 × 480	0.18 (22%)	0.18 (12%)	0.18 (6%)
640 × 480	0.17 (22%)	0.18 (11%)	0.18 (6%)
704 × 480	0.17 (22%)	0.18 (11%)	0.18 (6%)
720 × 400	0.17 (22%)	0.18 (11%)	0.18 (6%)
720 × 480	0.17 (22%)	0.18 (11%)	0.18 (6%)
800 × 600	0.17 (22%)	0.18 (11%)	0.18 (6%)
832 × 624	0.17 (22%)	0.18 (11%)	0.18 (6%)
1024 × 768	0.17 (21%)	0.18 (11%)	0.18 (6%)
1152 × 864	0.17 (22%)	0.18 (11%)	0.18 (6%)
1280 × 720	0.17 (21%)	0.18 (11%)	0.18 (6%)
1280 × 1024	0.17 (22%)	0.18 (11%)	0.18 (6%)
1600 × 1200	0.17 (21%)	0.17 (11%)	0.17 (5%)
1800 × 1440	0.17 (21%)	0.18 (11%)	0.18 (6%)
1920 × 1080	0.17 (22%)	0.18 (11%)	0.18 (6%)
1920 × 1200	0.17 (21%)	0.18 (11%)	0.18 (6%)
Median	0.17 (22%)	0.18 (11%)	0.18 (6%)
Mean	0.19 (24%)	0.20 (12%)	0.20 (6%)
Standard deviation	0.06 (8%)	0.07 (5%)	0.07 (2%)

Figure A.19: Load and store bandwidth for randomized image access pattern. The mean data for these cases are the third lines on the graphs in Figure 6.22. Inner loop is unrolled twice and utilizes branch delay slots. Index array is 256-bit aligned. Sub-banks are fixed at 4. Lanes are fixed at 4. Address generation resources are varied.

Address generators	**4**	**8**	**16**
Peak Bandwidth	**0.8 GB/s**	**1.6 GB/s**	**3.2 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.48 (60%)	0.56 (35%)	0.56 (18%)
176 × 144	0.47 (59%)	0.55 (35%)	0.55 (17%)
352 × 240	0.41 (52%)	0.46 (29%)	0.46 (14%)
352 × 288	0.40 (50%)	0.44 (28%)	0.44 (14%)
352 × 480	0.39 (49%)	0.43 (27%)	0.43 (13%)
480 × 480	0.38 (48%)	0.42 (26%)	0.42 (13%)
512 × 384	0.38 (47%)	0.42 (26%)	0.42 (13%)
544 × 480	0.38 (48%)	0.42 (26%)	0.42 (13%)
640 × 480	0.37 (46%)	0.40 (25%)	0.40 (13%)
704 × 480	0.37 (47%)	0.41 (25%)	0.41 (13%)
720 × 400	0.37 (47%)	0.41 (26%)	0.41 (13%)
720 × 480	0.37 (46%)	0.41 (25%)	0.41 (13%)
800 × 600	0.37 (46%)	0.40 (25%)	0.40 (13%)
832 × 624	0.37 (46%)	0.40 (25%)	0.40 (13%)
1024 × 768	0.36 (45%)	0.39 (24%)	0.39 (12%)
1152 × 864	0.36 (46%)	0.40 (25%)	0.40 (12%)
1280 × 720	0.36 (45%)	0.39 (25%)	0.39 (12%)
1280 × 1024	0.37 (46%)	0.40 (25%)	0.40 (12%)
1600 × 1200	0.35 (44%)	0.38 (24%)	0.38 (12%)
1800 × 1440	0.36 (45%)	0.39 (24%)	0.39 (12%)
1920 × 1080	0.36 (45%)	0.39 (24%)	0.39 (12%)
1920 × 1200	0.36 (44%)	0.39 (24%)	0.39 (12%)
Median	0.37 (46%)	0.40 (25%)	0.40 (13%)
Mean	0.38 (48%)	0.42 (26%)	0.42 (13%)
Standard deviation	0.03 (4%)	0.05 (3%)	0.05 (1%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.47 (59%)	0.55 (34%)	0.55 (17%)
176 × 144	0.46 (58%)	0.53 (33%)	0.53 (17%)
352 × 240	0.26 (33%)	0.27 (17%)	0.27 (9%)
352 × 288	0.25 (31%)	0.26 (16%)	0.26 (8%)
352 × 480	0.24 (30%)	0.25 (16%)	0.25 (8%)
480 × 480	0.24 (30%)	0.25 (16%)	0.25 (8%)
512 × 384	0.23 (29%)	0.24 (15%)	0.24 (8%)
544 × 480	0.24 (30%)	0.25 (16%)	0.25 (8%)
640 × 480	0.23 (29%)	0.24 (15%)	0.24 (8%)
704 × 480	0.23 (29%)	0.24 (15%)	0.24 (8%)
720 × 400	0.23 (29%)	0.24 (15%)	0.24 (8%)
720 × 480	0.23 (29%)	0.24 (15%)	0.24 (8%)
800 × 600	0.23 (29%)	0.24 (15%)	0.24 (7%)
832 × 624	0.23 (29%)	0.24 (15%)	0.24 (8%)
1024 × 768	0.22 (28%)	0.23 (14%)	0.23 (7%)
1152 × 864	0.23 (28%)	0.24 (15%)	0.24 (7%)
1280 × 720	0.22 (28%)	0.23 (15%)	0.23 (7%)
1280 × 1024	0.23 (28%)	0.23 (15%)	0.23 (7%)
1600 × 1200	0.22 (27%)	0.23 (14%)	0.23 (7%)
1800 × 1440	0.22 (27%)	0.23 (14%)	0.23 (7%)
1920 × 1080	0.22 (28%)	0.23 (15%)	0.23 (7%)
1920 × 1200	0.22 (27%)	0.23 (14%)	0.23 (7%)
Median	0.23 (29%)	0.24 (15%)	0.24 (8%)
Mean	0.25 (32%)	0.27 (17%)	0.27 (8%)
Standard deviation	0.07 (9%)	0.09 (6%)	0.09 (3%)

Figure A.20: Load and store bandwidth for randomized image access pattern. The mean data for these cases are the fourth lines on the graphs in Figure 6.22. Inner loop is unrolled twice and utilizes branch delay slots. Index array is 256-bit aligned. Sub-banks are fixed at 8. Lanes are fixed at 4. Address generation resources are varied.

Address generators	**4**	**8**	**16**
Peak Bandwidth	**0.8 GB/s**	**1.6 GB/s**	**3.2 GB/s**
Image Size	**Load Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.48 (60%)	0.56 (35%)	0.56 (18%)
176 × 144	0.48 (59%)	0.56 (35%)	0.56 (17%)
352 × 240	0.46 (57%)	0.52 (33%)	0.52 (16%)
352 × 288	0.45 (56%)	0.52 (32%)	0.52 (16%)
352 × 480	0.45 (56%)	0.51 (32%)	0.51 (16%)
480 × 480	0.44 (55%)	0.49 (31%)	0.49 (15%)
512 × 384	0.44 (54%)	0.50 (31%)	0.50 (16%)
544 × 480	0.43 (54%)	0.49 (31%)	0.49 (15%)
640 × 480	0.42 (53%)	0.47 (30%)	0.47 (15%)
704 × 480	0.43 (53%)	0.48 (30%)	0.48 (15%)
720 × 400	0.42 (53%)	0.47 (30%)	0.47 (15%)
720 × 480	0.42 (53%)	0.48 (30%)	0.48 (15%)
800 × 600	0.42 (52%)	0.47 (29%)	0.47 (15%)
832 × 624	0.42 (52%)	0.47 (29%)	0.47 (15%)
1024 × 768	0.41 (51%)	0.46 (28%)	0.46 (14%)
1152 × 864	0.41 (52%)	0.46 (29%)	0.46 (14%)
1280 × 720	0.41 (51%)	0.45 (28%)	0.45 (14%)
1280 × 1024	0.41 (52%)	0.46 (29%)	0.46 (14%)
1600 × 1200	0.40 (50%)	0.44 (28%)	0.44 (14%)
1800 × 1440	0.40 (50%)	0.45 (28%)	0.45 (14%)
1920 × 1080	0.41 (51%)	0.45 (28%)	0.45 (14%)
1920 × 1200	0.40 (50%)	0.45 (28%)	0.45 (14%)
Median	0.42 (53%)	0.47 (30%)	0.47 (15%)
Mean	0.43 (53%)	0.48 (30%)	0.48 (15%)
Standard deviation	0.02 (3%)	0.03 (2%)	0.03 (1%)
Image Size	**Store Bandwidth in GB/s (Percentage of Peak)**		
128 × 96	0.48 (60%)	0.56 (35%)	0.56 (18%)
176 × 144	0.48 (59%)	0.56 (35%)	0.56 (17%)
352 × 240	0.36 (45%)	0.39 (24%)	0.39 (12%)
352 × 288	0.34 (43%)	0.37 (23%)	0.37 (12%)
352 × 480	0.32 (40%)	0.34 (21%)	0.34 (11%)
480 × 480	0.31 (38%)	0.33 (20%)	0.33 (10%)
512 × 384	0.31 (39%)	0.33 (21%)	0.33 (10%)
544 × 480	0.31 (39%)	0.33 (21%)	0.33 (10%)
640 × 480	0.30 (38%)	0.32 (20%)	0.32 (10%)
704 × 480	0.30 (38%)	0.32 (20%)	0.32 (10%)
720 × 400	0.30 (38%)	0.32 (20%)	0.32 (10%)
720 × 480	0.30 (37%)	0.32 (20%)	0.32 (10%)
800 × 600	0.29 (36%)	0.31 (19%)	0.31 (10%)
832 × 624	0.29 (37%)	0.31 (19%)	0.31 (10%)
1024 × 768	0.28 (35%)	0.30 (19%)	0.30 (9%)
1152 × 864	0.29 (36%)	0.30 (19%)	0.30 (9%)
1280 × 720	0.28 (35%)	0.30 (19%)	0.30 (9%)
1280 × 1024	0.29 (36%)	0.30 (19%)	0.30 (9%)
1600 × 1200	0.28 (35%)	0.29 (18%)	0.29 (9%)
1800 × 1440	0.28 (35%)	0.29 (18%)	0.29 (9%)
1920 × 1080	0.28 (35%)	0.30 (18%)	0.30 (9%)
1920 × 1200	0.28 (35%)	0.29 (18%)	0.29 (9%)
Median	0.30 (37%)	0.32 (20%)	0.32 (10%)
Mean	0.32 (39%)	0.34 (21%)	0.34 (11%)
Standard deviation	0.06 (7%)	0.07 (5%)	0.07 (2%)

Figure A.21: Load and store bandwidth for randomized image access pattern. The mean data for these cases are the fifth lines on the graphs in Figure 6.22. Inner loop is unrolled twice and utilizes branch delay slots. Index array is 256-bit aligned. Subbanks are fixed at 16. Lanes are fixed at 4. Address generation resources are varied.

References

[1] Advanced Television Systems Committee. ATSC Digital Television standard. `http://www.atsc.org/Standards/A53`, September 16, 1995.

[2] Asanović, K. *Vector Microprocessors.* PhD thesis, Computer Science Division, University of California – Berkeley, 1998.

[3] City Plaza. C-Phone H-324 TV Set-Top VideoPhone specifications. `http://www.cityplaza.co.kr/com_pd14.htm`.

[4] Diefendorff, K., and Dubey, P. How multimedia workloads will change processor design. *IEEE Computer* (September 1997).

[5] Digital Television glossary. `http://www.DigitalTelevision.com/dtvbook/glossaryf.htm`.

[6] Dubey, P. Architectural and design implications of mediaprocessing. `http://www.research.ibm.com/people/p/pradeep/media_tutorial/ppframe.htm%`.

[7] E-commerce Webopedia, entry for "video standards". `http://e-comm.webopedia.com/TERM/v/video_standards.html`.

[8] Fromm, R. Vector IRAM performance modeling: `vsim-p` – The VIRAM performance simulator. Tech. rep., University of California – Berkeley, 1999. In progress.

[9] Hennessy, J., and Patterson, D. *Computer Architecture: A Quantitative Approach*, second ed. Morgan Kaufman, 1996.

[10] INHA University Multimedia Lab. Image coding. `http://www.ee.inha.ac.kr/~image/codingEng.html`.

[11] Home site of the JPEG and JBIG committees. `http://www.jpeg.org/`.

[12] JPEG FAQ. `http://www.faqs.org/faqs/jpeg-faq/`.

[13] Kane, G., and Heinrich, J. *MIPS RISC Architecture.* Prentice Hall, 1992.

[14] Kozyrakis, C. Vector IRAM microarchitecture manual. Tech. rep., University of California – Berkeley, 1999. In progress.

[15] Kozyrakis, C., Fromm, R., Martin, D., Asanović, K., and Patterson, D. A media-enhanced vector architecture for embedded memory systems. June 1999.

[16] Kozyrakis, C., and Patterson, D. A new direction for computer architecture research. *IEEE Computer 31*, 11 (November 1998), 24–32.

[17] Kozyrakis, C., Perissakis, S., Patterson, D., Anderson, T., Asanović, K., Cardwell, N., Fromm, R., Golbus, J., Bribstad, B., Keeton, K., Thomas, R., Treuhaft, N., and Yelick, K. Scalable processors in the billion-transistor era: IRAM. *IEEE Computer 30*, 9 (September 1997), 75–8.

[18] Martin, D. Vector extensions to the MIPS-IV instruction set architecture. `http://iram.cs.berkeley.edu/isa.ps`, 1999.

[19] Mayer-Patel, K. Personal communication, 1999.

[20] McCalpin, J. Stream "standard" results. `http://www.cs.virginia.edu/stream/standard/Bandwidth.html`, July 6, 1999.

[21] McVoy, L., and Staelin, C. lmbench: Portable tools for performance analysis. In *Proceedings of the USENIX 1996 Annual Technical Conference* (San Diego, CA, January 22-26, 1996), pp. 279–94.

[22] MPEG frequently asked questions with answers. `http://www.bmrc.berkeley.edu/research/mpeg/mpegfaq.html`.

[23] MPEG pointers and resources. `http://www.mpeg.org/MPEG/`.

[24] RSI Systems, Inc., Videoconferencing systems. ERIS specifications. `http://www.squarenet.com/rsi/erisspec.htm`.

[25] Sony. Multiscan 17seII/20seII. Operating instructions.

[26] Video Electronics Standards Association. `http://www.vesa.org/`.

Acknowledgements

The performance simulator `vsim-p` is built with the MINT+ toolkit for architecture simulation developed by Jack Veenstra of Sandcraft and extended for the VIRAM ISA by David Martin. This work could not have been possible without the contributions of Christoforos Kozyrakis, who has developed much of the VIRAM-1 microarchitecture, and Krste Asanović of MIT, whose Ph.D. thesis has been the source of many of the ideas for VIRAM.

I would also like to thank my advisor, David Patterson, for his support, guidance, and inspiration.