

Organizational Design Trade-Offs at the DRAM, Memory Bus, and Memory Controller Level: Initial Results

Vinodh Cuppu and Bruce Jacob
Electrical & Computer Engineering
University of Maryland, College Park
{ramvinod,blj}@eng.umd.edu

ABSTRACT

This paper presents initial results in a study of organization-level parameters associated with the design of the primary memory system—the DRAM system beneath the lowest level of the cache hierarchy. These parameters are orthogonal to architecture-level parameters such as DRAM core speed, bus arbitration protocol, etc. and include bus width, bus speed, number of independent channels, degree of banking, read burst width, write burst width, etc.; this study presents the effective cross-product of varying each of these parameters independently. The simulator is based on SimpleScalar 3.0a and models a fast (simulated as 2GHz), highly aggressive out-of-order uniprocessor. The interface to the primary memory system is fully non-blocking, supporting up to 32 outstanding misses at both the level-1 and level-2 caches.

Our simulations show the following: (a) the choice of primary memory-system organization is critical, as it can effect total execution time by a factor of 3x for a constant CPU organization and DRAM speed; (b) the most important factors in the performance of the primary memory system are the channel speed (bus cycle time) and the granularity of data access, the burst width—each of these can independently affect total execution time by a factor of 2x; (c) for small bursts, multiple narrow independent channels to the memory system exhibit better performance than a single wide channel; for large bursts, channel cycle time is the most important factor; (d) the degree of DRAM multi-banking plays a secondary role in its impact on total execution time; (e) the optimal burst width tends to be high (large enough to fetch an L2 cache block in 2 bursts) and scales with the block size of the level 2 cache; and (f) the memory queue sizes can be extremely large, due to the bursty nature of references to the primary memory system and the promotion of reads ahead of writes. Among other things, we conclude that the scheduling of the memory bus is the primary bottleneck and that it should be the focus of further study.

1 INTRODUCTION

The expanding performance gap between processor speeds and primary memory speeds has prompted a number of studies in DRAM systems. These studies range from memory-controller design [13, 12, 16, 4, 7] to integrating the DRAM core with the processor core for improved memory bandwidth and power consumption [3, 14, 10, 6, 9]. Additionally,

our recent DRAM study compares the performance of several contemporary DRAM architectures, including FPM, EDO, Synchronous, Enhanced Synchronous, SLDRAM, Rambus, and Direct Rambus [5]; one of its primary conclusions was that present bus architectures are becoming a bottleneck.

As a result, we have been studying bus and memory-controller organizations and have developed a simulation framework for placing disparate DRAM architectures on the same footing. The model defines a continuum of design choices that includes most contemporary DRAM architectures such as Rambus, Direct Rambus, PC-100/133/266 SDRAM, etc. Using this framework, we have investigated the organizational parameters of memory systems such as bus width, bus speed, number of independent channels, logical organization of channels, degree of banking, degree of interleaving, burst-mode vs. packetized access, read burst width, write burst width, split-transaction vs. pipelined buses, symmetric vs. asymmetric read/write request shapes, etc. We label these as “organizational” parameters because they are design choices that can be made independently of the architecture of the DRAM core.

In this paper, we present the simulation framework and an initial study of different organization-level parameters including bus speed, bus width, number of independent channels, degree of banking, and read/write burst width; despite the large range covered in this study, it really only begins to explore the space of memory-system organizations. We model a high-performance uniprocessor system (2GHz out-of-order superscalar CPU with lockup-free L1 and L2 caches [11]) and use the more memory-intensive applications in the SPEC’95 integer suite. In this study we ask and answer the following questions (clearly, our results and conclusions are dependent on our system configuration and choice of benchmarks):

- How important are the design choices made at the organization level of the primary memory system?

Holding constant the CPU architecture, the L1/L2 cache organizations, the DRAM architecture, and the DRAM speed, the choices made at the organization level can affect *total execution time* by a factor of 3x. The choices of memory-system organization can affect the memory overhead by a factor of 10x, but much of this overhead is hidden behind program execution. Clearly, the choices of organization are extremely important.

- What are the most significant organizational parameters that affect performance of the primary memory system?

Holding other factors constant, the read/write burst width¹ (the granularity of data access) can be responsible for differences in total execution time of 3x; the cycle time of the memory channel can be responsible for a factor of 2x; the number of independent channels connecting the CPU to the DRAMs can be responsible for a performance change of 25%. Other parameters are responsible for differences in total execution time of less than 15%.

- How does the degree of banking affect performance?

Surprisingly, the degree of banking has little impact on total execution time. While the memory-system overhead can decrease 10-20% by increasing the number of banks per channel beyond 1, much of the improvement is hidden behind CPU execution. The net result is a 5% improvement in total execution time.

- What are the performance trade-offs between the number of independent channels, the channel width, the channel speed, and the total system bandwidth (number of channels \times channel width \times channel speed)?

As one might guess, the total per-channel bandwidth (bus width \times bus speed) is often more important than the choice of either bus width or bus speed, because it takes the same amount of time to send 128 bits down a 16-bit, 800MHz channel as a 128-bit, 100MHz channel.

However, there are counterexamples. Whereas, for a given burst size, performance is not particularly sensitive to bandwidth, it is very sensitive to channel width or speed: for a given burst size, doubling the memory system's bandwidth can occasionally *increase* execution time, while changing the number of channels, the speed of a channel, or the width of a channel (and at the same time holding bandwidth constant) can often reduce total execution time by a significant amount.

We also make the following observations. First, and most importantly, there is a very complex tradeoff between the optimal burst size and the optimal system bandwidth configuration (number of channels, channel width, channel speed). The optimal burst size is wide enough to fetch an L2 cache block in two requests (e.g. 64-byte burst for a 128-byte L2 block size). Given a fixed burst size, the optimal choice of system bandwidth configuration changes dramatically from large burst sizes to small burst sizes: for example, what is good for large bursts (few independent channels) is the worst choice for small bursts, and what is good for small bursts (many independent channels) is the worst choice for large bursts. Because the interactions between system configuration and burst size can affect system performance by up to a factor of three, it is critically important to design the entire memory system to fit together—no one component of the memory system can be optimized in isolation. Given that the

1. Note that this term does not imply that the model is a burst-mode model. The term refers to the granularity of data access; for example, Direct Rambus has a packetized DRAM interface, as opposed to burst-mode DRAMs such as SDRAM or ESDRAM. However, its granularity of access is 128 bits (16 bytes). Thus, it would be modeled as having a 16-byte burst width.

optimal burst width scales with the level 2 cache block size, even the organization of the caches must play a role in the design of the primary memory system.

Second, the large degrees of internal banking in many of today's high-performance DRAMs (e.g. 16 banks in Direct Rambus DRAM), while perhaps necessary from an implementation standpoint, might be unnecessary from a performance standpoint. For the benchmarks studied, relatively low degrees of internal banking—in the range of 2x to 4x—are all that is necessary to achieve good performance.

Last, we did not place any restrictions on the size of the memory controller's request queue. Given that the combination of an 8-byte burst and a 128-byte cache block produces 16 requests per L2 read miss, a system with 32 MSHRs can have up to 512 outstanding requests in the memory system. For medium and large burst sizes, we saw relatively small queue sizes (up to tens of entries, down to 1 or 0 on average). By contrast, for small burst sizes, we frequently saw queue lengths in the tens of thousands, which is due to the fact that write requests can be stalled for arbitrarily long periods of time if a string of read requests appears. Future work will look at the effects of a finite queue size.

As previously mentioned, one of the primary results from our prior work was that present bus architectures are becoming a bottleneck. This study comes to the same conclusion. Our observations that small bursts require multiple independent channels for good performance suggest the interleaving of small bursts on a single channel to be expensive. Our observations that the memory queue lengths are enormous for small bursts suggest that interleaving small bursts creates bus traffic jams. Our observations that channel speed can be more important than channel bandwidth suggest that two different configurations with equal bandwidth do not necessarily exploit that bandwidth with the same degree of efficiency. These results all point to bus scheduling as the bottleneck. Future work will be to investigate this more closely.

2 SIMULATION FRAMEWORK & EXPERIMENTAL METHODOLOGY

2.1 High-Performance Memory Systems Primer, Briefly

High-performance memory systems are not structured as if each DRAM is connected directly to the CPU; there are usually several layers of memory controllers that serve to reduce the amount of time spent on an address or data bus. Typically, there is a memory controller ASIC that is integrated onto the DIMM itself that performs the \overline{RAS} and \overline{CAS} commands—what is usually called “the memory controller” is only responsible for scheduling requests to the DIMMs over the memory channel; the controller does not usually control the DRAMs directly. This enables a memory system to have several independent banks that can be active at the same time, enabling relatively full utilization of the data bus, even though the time it takes to get data out of the DRAM core is far longer than the bus transmission time. If there were only one bank per memory channel, there could be no such overlap, and the fastest rate at which requests could be serviced would be the time to pull data from the DRAM core. For more information, see [1, 8, 15].

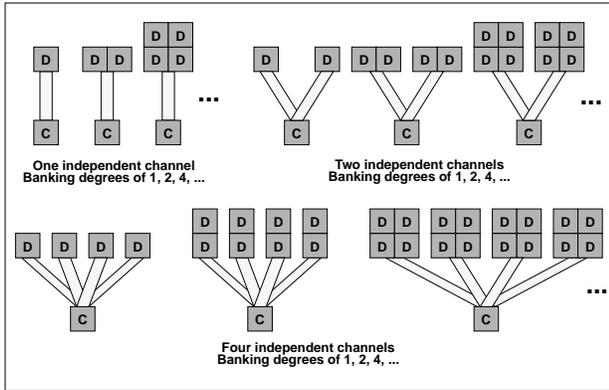


Figure 1: Channels and banks. This study looks at varying such parameters as the number of independent channels and the number of independent DRAM banks attached to each channel. C = CPU, D = DRAM bank.

2.2 Channels and Banks

The fundamental idea in this work is to define a model for the primary memory system that represents most DRAM organizations in existence, including burst-mode organizations such as SDRAM and packetized organizations such as Rambus (these being the two primary competing commercial standards), as well as almost everything else in between.

Several example memory-system organizations that can be represented by our model are illustrated in Figure 1. A single DRAM device can handle one request at a time and produces a certain number of bits per request: this is the device-level transfer width. DRAM devices are ganged together into banks, each of which is independent and can service a different request than all other banks at any given moment. The bank is the smallest unit of granularity represented in this model. Whether a “bank” is a single physical device or a sub-component within a single physical device need not be specified. A single bank has a transfer width at least as wide as the data bus. Each channel is a split-transaction address-bus/data-bus pair and is connected to potentially multiple banks, each of which is operated independently of the others; using multiple banks per channel supports concurrent transactions at the channel level. The CPU connects via an on-board memory controller to potentially multiple channels, each of which is operated independently of the others; using multiple channels supports concurrent transactions at the DRAM subsystem level. The bit mapping from address to channel/bank/row attempts to best exploit the available concurrency in the physical organization by assigning the lowest-order bits (which change the most frequently) to the channel number, the next bits to the bank number, etc. Counters in our simulation results show that the requests are divided evenly across the channels in a system and across the banks in each channel.

This is a very simple organization that accounts for most existing DRAM architectures: clearly, it can emulate organizations such as PC-XXX SDRAM, but it can also emulate Rambus-style organizations by increasing the degree of banking and scaling the channel width and speed, as Rambus devices use normal DRAM cores and are banked internally.

For the studies presented in this paper, we did not explore all possible combinations of channel speed and channel width to obtain the same bandwidth. For example, as shown in Fig-

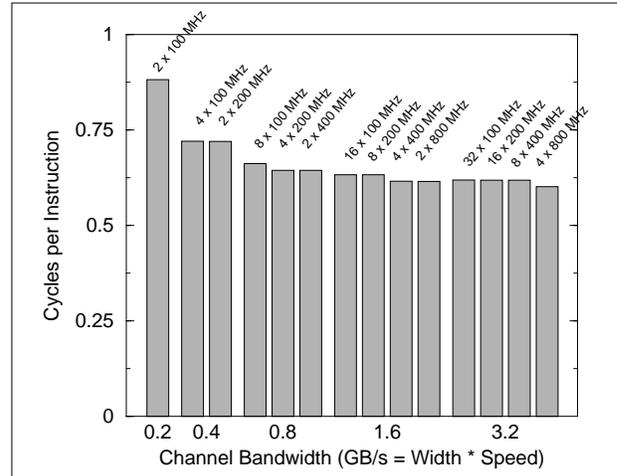


Figure 2: Performance as a function of bus width and bus speed.

Though there is up to a 5% difference between different combinations of bus width and bus speed that yield the same bandwidth, we cut the number of combinations simulated to reduce simulation time.

ure 2, there is a 5% performance range between a 1byte bus running at 800 MHz vs. a 2byte bus at 400MHz vs. a 4byte bus at 200MHz vs. an 8byte bus at 100MHz, with the highest frequency bus yielding the best performance. To reduce the number of simulations run for this paper we simulated the following combinations: 1x200, 1x400, 1x800, 2x800, 4x800, 8x800 (bandwidths from 200MB/s to 6400MB/s).

2.3 Burst Timing

For the DRAM core speed, we use parameters from the latest SDRAM, which has reasonably fast timing specifications and is common to PC-100 and Direct Rambus designs. This gives us the read and write bus and bank occupancies shown in Figure 3, which are similar to those reported in the literature [1, 8, 15]. The figure presents numbers for burst widths equal to the data bus width, twice the bus width, and four times the bus width. A burst is the smallest atomic transaction size—all read and writes requests are processed as an integral number of bursts, and the bursts of different requests may be multiplexed in time over the same channel. We model the bus turn-around time as a constant number of bus cycles; for this study, we used 1 cycle.

Note that this interface model covers burst-mode DRAM architectures such as SDRAM, ESDRAM, and burst-mode SLDRAM, and it also covers packetized DRAM architectures such as Rambus, Direct Rambus, and packetized SLDRAM. The only difference with moving to a packetized interface is that the address bus packet scales with the data bus packet in the length of time it occupies the address bus. Since the two are scheduled together, there is no additional overhead imposed by this scheme.

2.4 Burst Ordering

If a burst is smaller than the level-2 cache line size, then there are a number of options for the ordering of the burst-sized blocks that make up the request. In this study, the block containing the critical word is always fetched first and takes priority over any other block in the queue, unless that block also

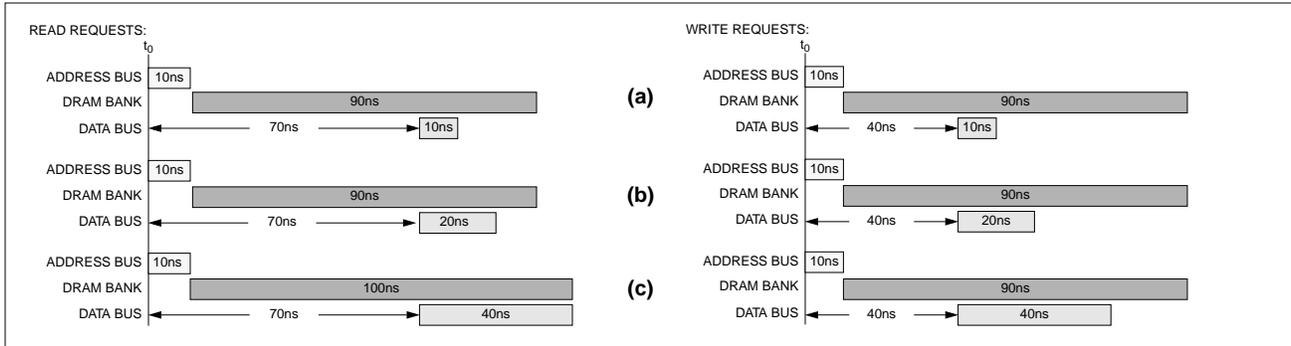


Figure 3: Bus and bank occupancies for 100MHz channel. Each DRAM request requires the address bus, the data bus, and whatever bank it is destined for. The shape of these request blocks is dependent on the burst widths. Figures are shown for burst-widths equal to (a) 1x the bus width, (b) 2x the bus width, and (c) 4x the bus width. One of the interesting points is that, though reads and writes are asymmetric, they become less so as the burst width increases.

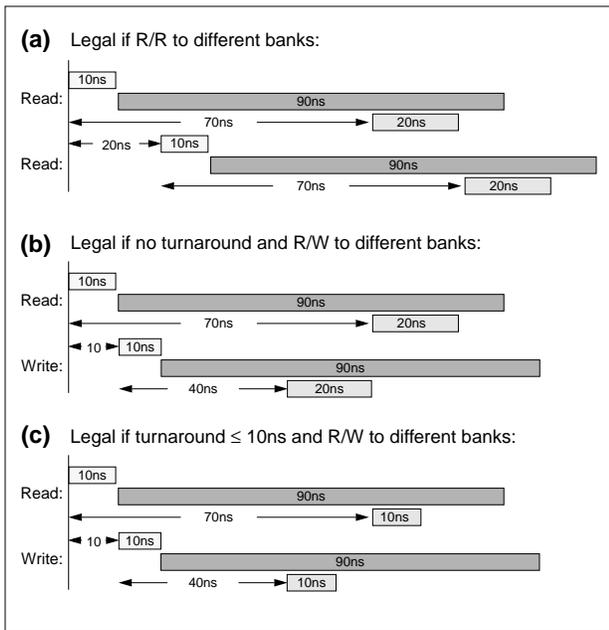


Figure 4: Concurrency within a single channel. If two concurrent reads require different banks, they can be pipelined across the address and data bus as shown in (a). Writes can be nested inside of reads, provided the bus turnaround time is low (b) or the burst width is small (c).

contains a critical word. Write requests are always given lowest priority and tend to stack up in the queue until all the reads drain from the queue.

2.5 Handling Concurrency

With multiple channels in a system, it is easy to see how concurrency can be exploited. However, within a single channel, provided that there is sufficient banking to support it, there can also be support for concurrency. Figure 4 illustrates several of the ways back-to-back requests are overlapped in time, sharing the common resources. Back-to-back reads can be pipelined, provided they require different banks. Back-to-back read/write pairs can be similarly pipelined, but it is also possible to nestle writes “inside of” reads, as shown in Figures 4(b) and (c), provided the conditions support it. This last feature is only possible because the asymmetric nature of read/write requests. Note that, though reads and writes are asymmetric, they look less so as the burst width increases and

the time that the data bus is held grows large. This will become important: it is more efficient to interleave symmetric requests, because there is less wasted dead time on the bus.

2.6 CPU Model

To obtain accurate timing of memory requests in a dynamically reordered instruction stream, we integrated our code into SimpleScalar 3.0a, an execution-driven simulator of an aggressive out-of-order processor [2]. Our simulated processor is eight-way superscalar; its simulated cycle time is 0.5ns (2GHz clock). Its L1 caches are split 64KB/64KB; both are 2-way set associative; both have 64-byte linesizes. Its L2 cache is unified 1MB, 4-way set associative, writeback, has a 128-byte linesize and a 10-cycle access time. The L1 and L2 caches are both lockup-free, and both allow up to 32 outstanding requests at a time. For our lockup-free cache model, a load instruction that misses the L2 cache is blocked until it obtains an MSHR, and it holds the MSHR *only until the critical burst of data returns* (remember that the atomic unit of transfer between the CPU and DRAM system is a burst). This scheme frees up the MSHR relatively quickly, allowing subsequent load instructions that miss the L2 cache to commence as soon as possible. This scheme is relatively expensive to implement, as it assumes that the cache tags can be checked for the subsequently arriving blocks without disturbing cache traffic. We model this optimization to put the highest possible pressure on the physical memory system—it represents the highest rate at which the processor can generate concurrent memory accesses given the number of available MSHRs.

2.7 Timing Calculations

Much of the DRAM access time is overlapped with instruction execution. To determine the degree of overlap, we run a second simulation with perfect primary memory (no overhead). Similar to the methodology in [5], we partition the total application execution time into three components: T_P , T_M and T_O which correspond to time spent processing, time spent stalling for memory, and the portion of time spent in the memory system that is successfully overlapped with processor execution. In this paper, time spent “processing” includes all activity above the primary memory system, i.e. it contains all processor execution time and L1 and L2 cache activity. Let T_{REAL} be the total execution time for the realistic simulation;

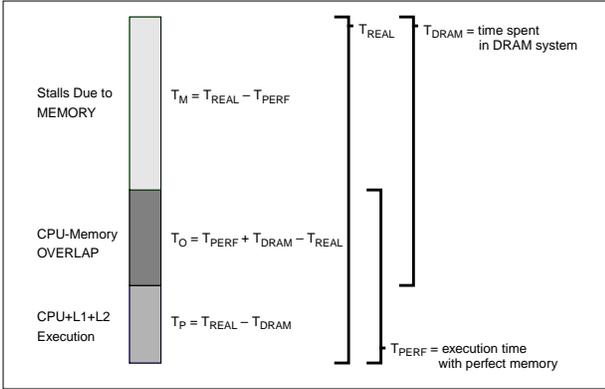


Figure 5: Definitions for execution-time breakdowns. The results of several simulations are used to show time spent in the memory system vs. time spent processing vs. the amount of memory latency hidden by the CPU.

let T_{PERF} be the execution time with a perfect DRAM system; let T_{DRAM} be the total time spent in the DRAM system. Then we have the following:

- $T_P = T_{\text{REAL}} - T_{\text{DRAM}}$
- $T_M = T_{\text{REAL}} - T_{\text{PERF}}$
- $T_O = T_{\text{PERF}} + T_{\text{DRAM}} - T_{\text{REAL}}$

The relationships between the different time parameters are illustrated in Figure 5.

3 EXPERIMENTAL RESULTS

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

- {1, 2, 4} independent channels
- {1, 2, 4} banks per channel
- {8, 16, 32, 64, 128} byte burst widths
- {1, 2, 4, 8} byte data-bus widths
- {200, 400, 800} MHz bus speeds (equivalent to 100, 200, 400 MHz dual data rate)
- {gcc, perl} from SPEC'95 known to have relatively large memory footprints

As described earlier, we did not simulate every combination of bus width and bus speed. The simulated L1/L2 cache line sizes are 64/128 bytes, and, for a few configurations, we also simulated L1/L2 linesizes of 32/64 bytes.

The following sections each present an analysis of a slightly different slice through the data. The unit of performance is cycles per instruction: a direct measurement of execution time, given a fixed cycle time and the length of each program. Note that for some system configurations (but not all), total execution time is further broken down into the components described in Section 2.7.

3.1 The Effects of Burst Width and Bandwidth

We begin by presenting in Figure 6 the total execution time as a function of both burst width and memory-system band-

width. On the x-axis is the system bandwidth, which is total channels \times channel width \times channel speed. For each bandwidth value, there are a number of configurations that represent different combinations of channels/width/speed. For each configuration, there are five stacked bars representing the total execution time for burst widths of 8, 16, 32, 64, and 128 bytes.

Among other things, the graphs show that for a given bandwidth configuration, the choice of burst size can affect execution time significantly—e.g., by a factor of just under 3x for gcc and just under 2x for perl. This clearly shows the importance of selecting an appropriate burst size. Though the optimal burst width depends on bandwidth and channel speed (optimal burst width is around 32 bytes for 200MHz channels, and around 64 bytes for 400 and 800MHz channels), it tends to be relatively large in general: for most configurations, it is 64 bytes. Figure 7 shows that it is also dependent on cache block size. The data are for a L2 cache block of size 64 bytes, and the graph shows the optimal burst width to be 32 bytes—i.e., the burst should be large enough to fetch a level-2 cache block in two requests.

In Figure 6, if one can ignore the noise, there is a gradual curve that slopes down as bandwidth increases, showing the effects of increased bandwidth on execution time. The slope reflects a 5–10% improvement in execution time for every doubling of memory-system bandwidth, which is far less significant than the effect that burst width has on performance. Within a fixed bandwidth class, the choice of bus speed and number of channels is significant, but not as significant as doubling or halving the bandwidth. For example, at 800MB/s, the effect of moving from a quad 200MHz 1-byte bus organization to a dual 400MHz 1-byte bus organization to a single 800MHz 1-byte bus organization yields a smaller performance difference than moving to a 400MB/s or 1.6GB/s organization.

In summary, burst width is an extremely significant parameter that overshadows both raw bandwidth and the details of how you choose your bandwidth (number of channels, channel width, channel speed).

3.2 Optimal Burst Width vs. Channel Organization

Next, we look more closely at optimal burst size in Figures 8 and 9. In each figure there are several graphs, each of which represents data for a constant burst width. Each graph depicts the total execution time (and for some bars, a break-down as well) for constant bitwidth organizations. Note that the data points at each bitwidth may have different bandwidths. At each data point, there are three vertical bars, corresponding to degrees of multibanking of 1, 2, and 4 banks per channel.

The graphs illustrate that there are three distinct regions of behavior, corresponding to small burst sizes, medium burst sizes, and large burst sizes. At small burst sizes (8 bytes), the parameter that influences performance the most is the number of independent channels: all 1-channel configurations have roughly the same performance; all 2-channel configurations have roughly the same performance; all 4-channel configurations have roughly the same performance—this despite the configuration's bandwidth. For a 32-bit datapath, the three configurations that are comprised of 4 8-bit channels all outperform the 2x16-bit 800MHz configuration by 12.5% and

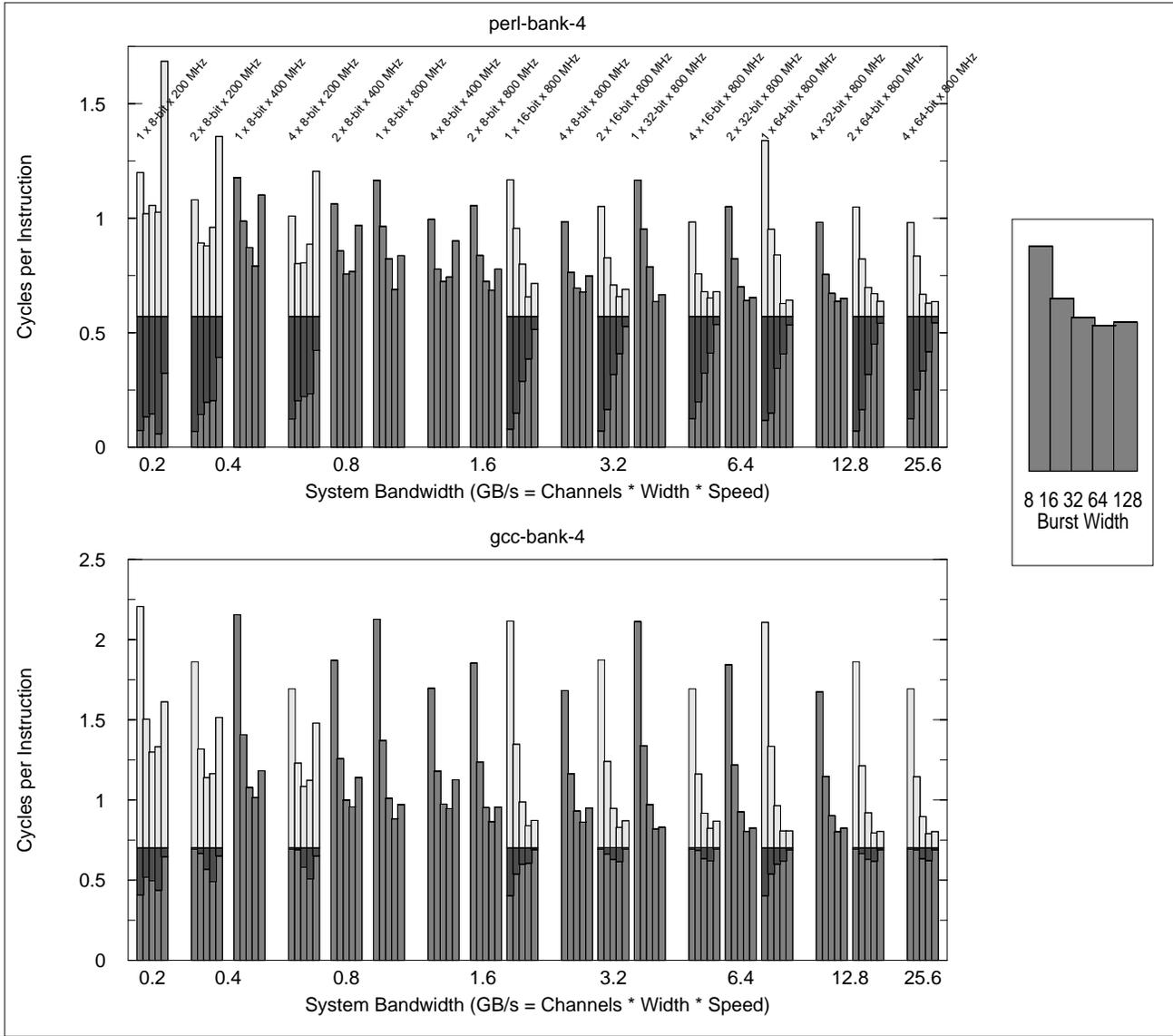


Figure 6: Bandwidth and burst width.

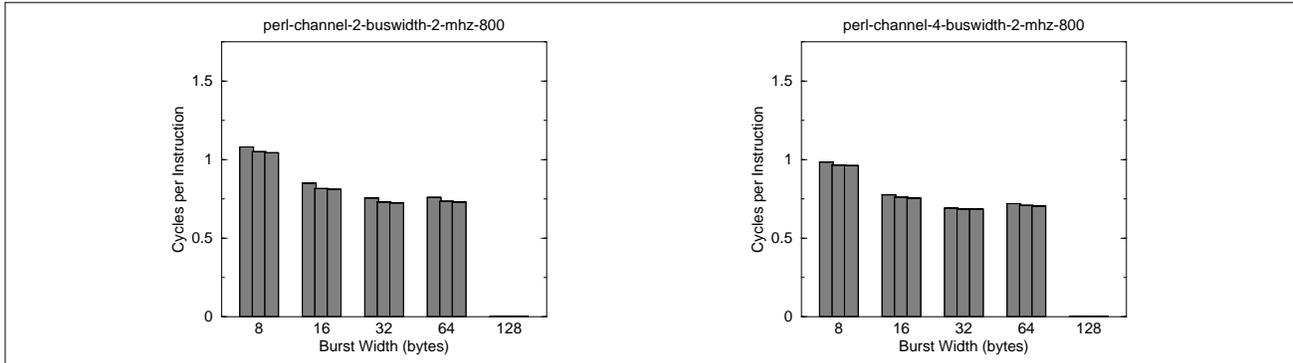


Figure 7: Optimal burst width for 32/64-byte L1/L2 line sizes. At each data point, there are three histograms representing the execution time as a function of the degree of banking. From left to right, the vertical bars show performance for 1, 2, and 4 banks per channel. There is no data for 128-byte burst, because such a burst size does not make sense for a 64-byte cache block. While the data in Figure 6 suggest the optimal burst width to be 64 bytes, this shows that the optimal burst size is 32 bytes when the L2 cache block is 64 bytes. Our conclusion is that the optimal burst width scales with the L2 cache size: it is large enough to fetch an L2 cache block in two requests.

the 1x32-bit 800MHz configuration by 25%. This happens even though the worse-performing configurations have 2x and 4x the bandwidth of the better-performing configura-

tions—e.g., the 4x8-bit 200MHz system has a bandwidth of 800MB/s and outperforms the 1x32-bit 800MHz system (which has 3.2GB/s bandwidth) by 25%. This suggests that

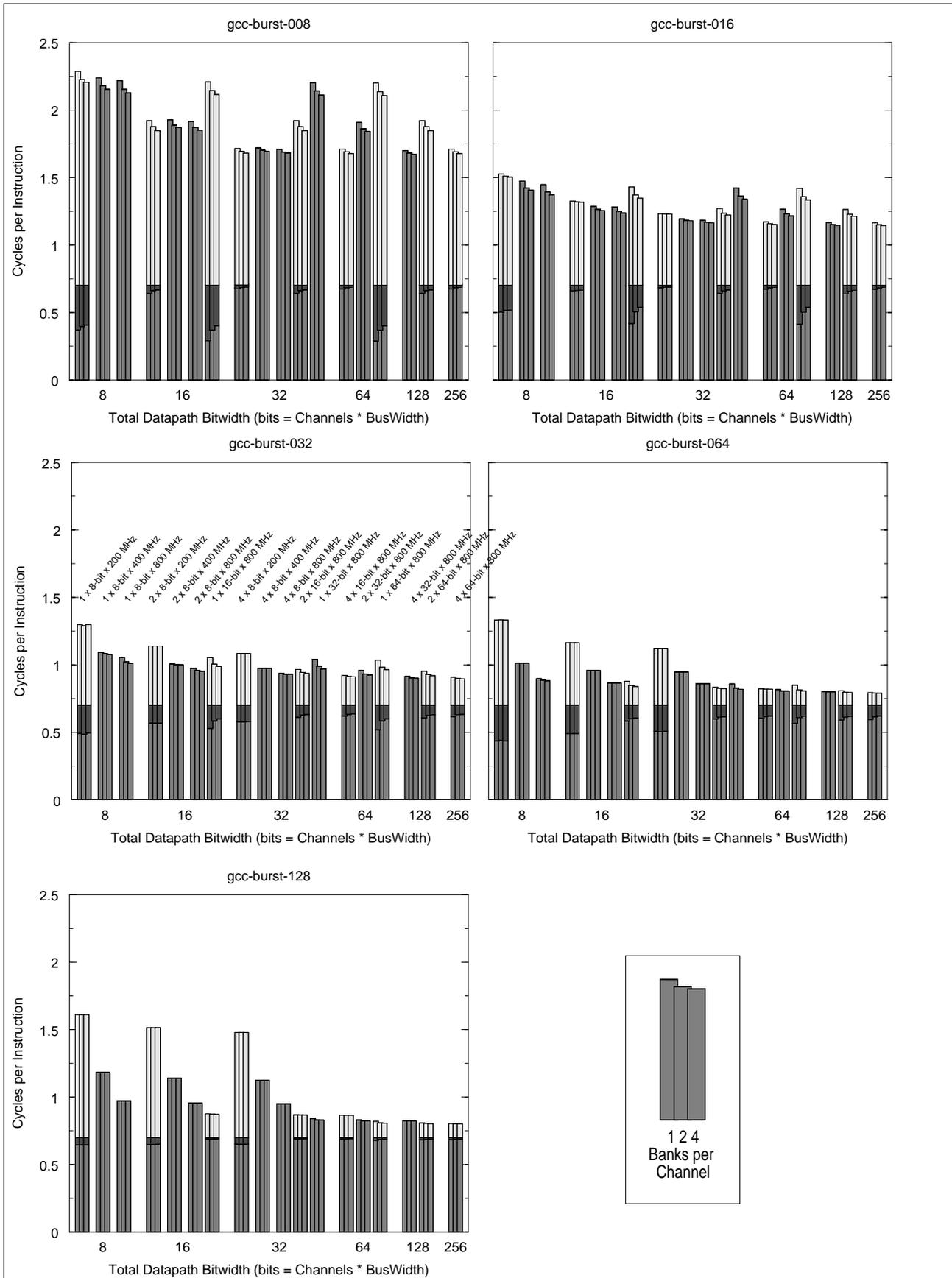


Figure 8: Burst width and channel organization tradeoffs — GCC.

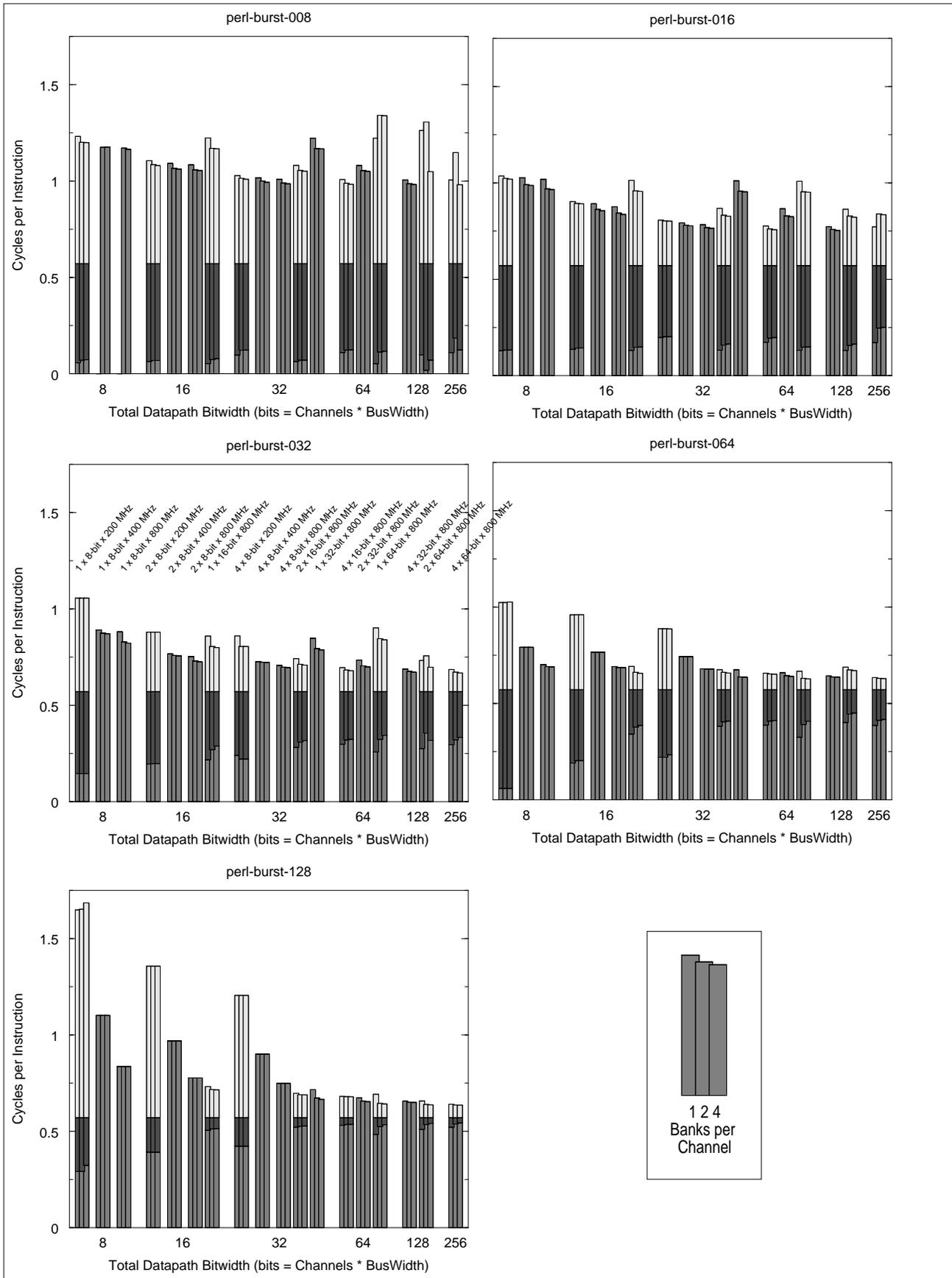


Figure 9: Burst width and channel organization tradeoffs — PERL.

further dividing the bitpath would yield further improvements—perhaps 8 4-bit channels would continue to yield improved performance. However, simply changing the burst width yields better results.

At medium burst sizes (32 bytes), there is little difference to be seen across all configurations. It is clear that the configurations with slower busses and narrower busses are likely to do slightly worse, but the difference between the best and worst configurations is roughly 25–30%.

At large burst sizes (128 bytes), it is no longer the case that more channels yields better performance; in fact, increasing the number of channels always degrades performance. For example, again at the 32-bit data point, the three configurations at 800MHz (all of which have identical bandwidth) show the effect of going from 4x8-bit to 2x16-bit to 1x32-bit configurations: in contrast to the behavior seen at small burst sizes, increasing the number of independent channels worsens performance. The most significant influence on performance for large burst sizes comes from the channel *speed*—note, for example, that the worst performance comes from 200MHz channels, which have roughly identical performance regardless of the bandwidth represented. The best performance comes from 800MHz channels, all of which perform within 10% of each other. At this burst width, simply increasing bandwidth makes little difference in execution time, provided the channel speed remains the same.

In summary, there is a delicate trade-off between the optimal burst size and the channel configuration: optimal choices in channel configuration (the number of channels, the speed of each channel, and the width of each channel) change dramatically depending on the choice of burst width. The optimal burst width appears to be somewhere between medium and large (64 bytes per burst), and we showed earlier that this parameter seems to scale with cache block size. Therefore, there are no blanket statements that cover memory-system design: each system must be optimized by taking into account all aspects of the design—no one component can be optimized in isolation.

3.3 A Closer Look at Banking and Burst Width

The graphs in Figure 10 illustrate the degree of memory overlap for several configurations. Some interesting things to note: first, with a single channel (top left column), gcc manages to overlap a fair amount of memory activity with CPU execution; as the number of independent channels increases, the system becomes much more streamlined, lowering the memory overhead rapidly. However, it also becomes more difficult for the system to overlap memory activity with CPU execution, as shown in the very small overlap components. Second, the perl benchmark does not have this problem—its behavior is such that it can always overlap a significant component of its memory activity with CPU execution. Clearly, this behavior is benchmark-dependent. Last, note the behavior of the 8-bit configuration (the bottom row of graphs). As we have pointed out before, as bus widths become narrow, large burst sizes tend to perform worse—this graph demonstrates that the problem occurs even earlier. By increasing the burst width from 16 bytes to 32 bytes, the memory overhead is almost always *increased*; often, this increase is hidden by CPU execution, but it is clear that there are two factors at

work: small bursts making it more difficult to use the memory system, and large bursts that occupy the busses for such a long duration that the average memory access is stalled waiting for resources.

The graphs show that the degree of banking has a noticeable impact on the total memory-system time, even though it might not translate to much in terms of total execution time. For instance, at 16-bit busses (the top two rows of graphs), each doubling of the number of banks decreases the overhead of the memory system by 10-20%. This ultimately translates to a net savings of around 5% in execution time due to the degree of overlap with CPU execution time.

4 CONCLUSIONS

We have found that the organization of the memory system is extremely important and can affect the total execution of the application by a factor of 3x. Unfortunately, there are no choices that are universally good—the interaction of the parameters is such that no component can be optimized individually. The only rules of thumb are that the optimal burst size scales with the L2 blocksize, and that faster channels are usually better.

As previously mentioned, one of the primary results from our prior work was that present bus architectures are becoming a bottleneck. This study comes to the same conclusion. The fact that small bursts require multiple independent channels for good performance suggests that the interleaving of small bursts on a single is expensive. Observations of the runtime lengths of the memory queues, which are enormous for small bursts, suggest that interleaving small bursts can create bus traffic jams. The fact that channel speed can be more important than channel bandwidth suggests that two different configurations with equal bandwidth do not necessarily exploit that bandwidth with the same degree of efficiency.

These results point to bus scheduling as the primary overhead. Possible explanations include intermingling writes with reads, yielding turnaround overhead and odd-shaped interleaved patterns (due to the asymmetric nature of reads & writes). Small bursts cause major backups in the memory system, because the time to transfer a burst is on the order of the bus turnaround overhead—and because the asymmetric nature of read requests vs. write requests makes it inefficient to interleave the two. For larger bursts, the turnaround time is amortized, and interleaving reads with writes is not much different than interleaving read pairs or write pairs, because the time to hold the data bus is extremely long.

More directions for future study include the use of symmetric read/write shapes to simplify bus scheduling, the effects of cache organizations (since block size has such a dramatic influence), the effects of turnaround time (maybe two separate data busses would do better), as well as the use of realistic queue sizes and conventional MSHR designs.

REFERENCES

- [1] W. R. Bryg, K. K. Chan, and N. S. Fiduccia. "A high-performance, low-cost multiprocessor bus for workstations and midrange servers." *The Hewlett-Packard Journal*, vol. 47, no. 1, February 1996.
- [2] D. Burger and T. M. Austin. "The SimpleScalar tool set, version 2.0." Tech. Rep. CS-1342, University of Wisconsin-Madison, June 1997.
- [3] D. Burger, J. R. Goodman, and A. Kagi. "Memory bandwidth limitations of future microprocessors." In *Proc. 23rd Annual International Symposium on Computer Architecture (ISCA'96)*, Philadelphia PA, May 1996, pp. 78–89.

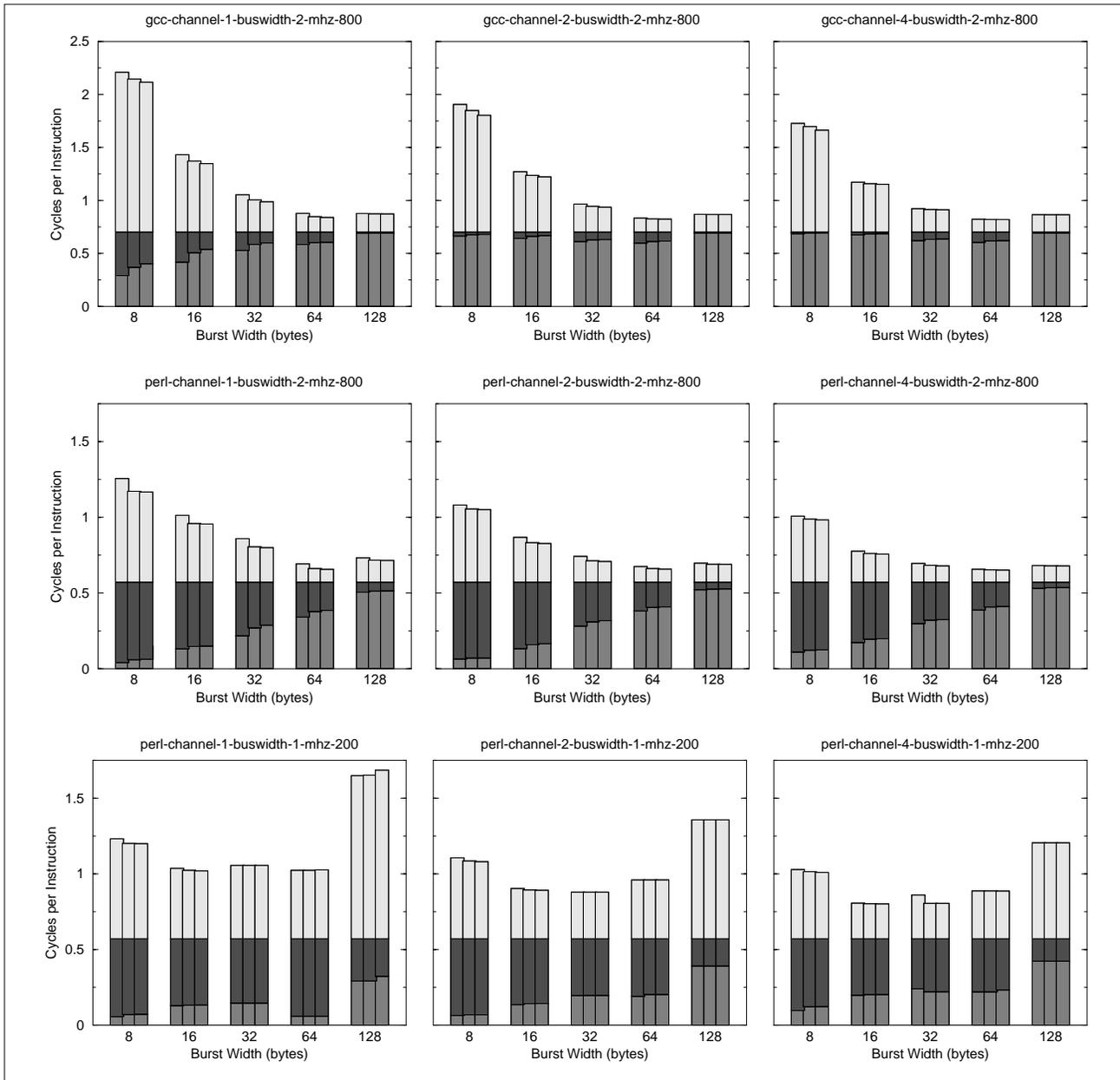


Figure 10: Banking degree and burst width. Each graph shows three histograms for each burst width: the three bars correspond to banking degrees of 1, 2, and 4 banks per channel.

- [4] J. Carter, W. Hsieh, L. Stoller, M. Swanson, L. Zhang, and et al. "Impulse: Building a smarter memory controller." In *Proc. Fifth International Symposium on High Performance Computer Architecture (HPCA'99)*, Orlando FL, January 1999, pp. 70–79.
- [5] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. "A performance comparison of contemporary DRAM architectures." In *Proc. 26th Annual International Symposium on Computer Architecture (ISCA'99)*, Atlanta GA, May 1999, pp. 222–233.
- [6] R. Fromm, S. Perissakis, N. Cardwell, C. Kozyrakis, B. McGaughy, D. Patterson, T. Anderson, and K. Yelick. "The energy efficiency of IRAM architectures." In *Proc. 24th Annual International Symposium on Computer Architecture (ISCA'97)*, Denver CO, June 1997, pp. 327–337.
- [7] S. I. Hong, S. A. McKee, M. H. Salinas, R. H. Klenke, J. H. Aylor, and W. A. Wulf. "Access order and effective bandwidth for streams on a Direct Rambus memory." In *Proc. Fifth International Symposium on High Performance Computer Architecture (HPCA'99)*, Orlando FL, January 1999, pp. 80–89.
- [8] T. R. Hotchkiss, N. D. Marschke, and R. M. McColsky. "A new memory system design for commercial and technical computing products." *The Hewlett-Packard Journal*, vol. 47, no. 1, February 1996.
- [9] K. Inoue, K. Kai, and K. Murakami. "Dynamically variable line-size cache exploiting high on-chip memory bandwidth of merged DRAM/logic LSIs." In *Proc. Fifth International Symposium on High Performance Computer Architecture (HPCA'99)*, Orlando FL, January 1999, pp. 218–222.
- [10] C. Kozyrakis, et al. "Scalable processors in the billion-transistor era: IRAM." *IEEE Computer*, vol. 30, no. 9, pp. 75–78, September 1997.
- [11] D. Kroft. "Lockup-free instruction fetch/prefetch cache organization." In *Proc. 8th Annual International Symposium on Computer Architecture (ISCA'81)*, Minneapolis MN, May 1981.
- [12] S. McKee, A. Aluwihare, B. Clark, R. Klenke, T. Landon, C. Oliver, M. Salinas, A. Szymkowiak, K. Wright, W. Wulf, and J. Aylor. "Design and evaluation of dynamic access ordering hardware." In *Proc. International Conference on Supercomputing*, Philadelphia PA, May 1996.
- [13] S. A. McKee and W. A. Wulf. "Access ordering and memory-conscious cache utilization." In *Proc. International Symposium on High Performance Computer Architecture (HPCA'95)*, Raleigh NC, January 1995, pp. 253–262.
- [14] A. Saulsbury, F. Pong, and A. Nowatzyk. "Missing the memory wall: The case for processor/memory integration." In *Proc. 23rd Annual International Symposium on Computer Architecture (ISCA'96)*, Philadelphia PA, May 1996, pp. 90–101.
- [15] R. C. Schumann. "Design of the 21174 memory controller for DIGITAL personal workstations." *Digital Technical Journal*, vol. 9, no. 2, pp. 57–70, 1997.
- [16] M. Swanson, L. Stoller, and J. Carter. "Increasing TLB reach using superpages backed by shadow memory." In *Proc. 25th Annual International Symposium on Computer Architecture (ISCA'98)*, Barcelona, Spain, June 1998, pp. 204–213.