Minimizing the Page Close Penalty: Indexing Memory Banks Revisited

Tomas Rokicki
Computer Systems Laboratory
HP Laboratories Palo Alto
HPL-97-18 (R.1)
June 16th , 2003*

memory controller, page-mode, indexing, addressing, performance, prefetching

This paper introduces several new techniques to optimize the use of page-mode and prefetching to improve the performance of a memory system. We start by improving the basic bank indexing scheme, allowing higher hit rates to be attained with a smaller number of banks. Next, we introduce the idea of keeping only a subset of the possible banks open. When this is done, only a small fraction of page-mode misses are to open banks, while the banks that are open capture the great majority of the possible page-mode and sequential prefetching hits. This both improves the performance of workloads with a high hit rate and significantly decreases the risk of page misses under difficult or random workloads. This idea might also simplify the memory controller. In our workloads, using only eight bank controllers on a memory system supporting only 32 banks, 55% of the all read references were satisfied by data already in the bus drivers, requiring no DRAM latency at all. We compare different techniques for mapping a smaller number of bank controllers onto a larger number of memory banks, and show that using a fully-associative mapping works best.

The report finishes with a number of auxiliary investigations. We show that there is a slight advantage to not associating the prefetch buffers with the bank controllers, but rather using a separate LRU prefetch cache. We show that DRAM refresh has a small impact on the results, while the cache line size has a large impact.

* Internal Accession Date Only
Approved for External Publication

Minimizing the Page Close Penalty: Indexing Memory Banks Revisited

Tomas Rokicki

January 2, 1997

Abstract. This paper introduces several new techniques to optimize the use of page mode and prefetching to improve the performance of a memory system. We start by improving the basic bank indexing scheme, allowing higher hit rates to be attained with a smaller number of banks. Next, we introduce the idea of keeping only a subset of the possible banks open. When this is done, only a small fraction of page-mode misses are to open banks, while the banks that are open capture the great majority of the possible page-mode and sequential prefetching hits. This both improves the performance of workloads with a high hit rate and significantly decreases the risk of page misses under difficult or random workloads. This idea might also simplify the memory controller. In our workloads, using only eight bank controllers on a memory system supporting only 32 banks, 55% of the all read references were satisfied by data already in the bus drivers, requiring no DRAM latency at all. We compare different techniques for mapping a smaller number of bank controllers onto a larger number of memory banks, and show that using a fully-associative mapping works best.

The report finishes with a number of auxilliary investigations. We show that there is a slight advantage to not associating the prefetch buffers with the bank controllers, but rather using a separate LRU prefetch cache. We show that DRAM refresh has a small impact on the results, while the cache line size has a large impact.

Keywords. memory controller, pagemode, indexing, addressing, performance, prefetching.

Contents

1 Introduction: The Page Mode Gamble Revisited **3**

2 Assumptions **4**

3 Memory Controller Description **4**

4 The Xor Bank Indexing Scheme **8**

5 Experiments **12**

6 Results and Analysis **12**

7 Alternative Bank Organizations **17**

8 Using a Prefetch Cache **20**

9 A Tough Workload **21**

10 The Impact of Refresh **24**

11 The Impact of Cache Line Size **24**

12 Discussion **25**

13 Acknowledgements **27**

14 References **27**

1 Introduction: The Page Mode Gamble Revisited

In the previous report on indexing memory banks [TR96], we discussed how a memory system should organize banks of memory to maximize the page mode hit rate. In this note, we extend that work slightly to show how the cost of keeping pages open in a page-mode memory controller can be minimized.

This report should only be read as an addendum to [TR96]; for brevity, we omit here the motivation and background material included in that report.

In a world of commodity components, where each computer vendor uses the same processors, caches, busses, and DRAM memories, designing an effective memory controller and memory system is one of the few key things that can be done to differentiate a product. The increasing importance of memory latency due to increasing processor clock rates and scalarity makes it even more critical to be competitive in this area. The techniques described in this and the previous report can combine to cut memory latency in half with only minor changes over a basic DRAM memory system. No additional chips are required; we simply add some modest buffering and control to what already needs to be there and attain a dramatic performance improvement.

In a DRAM memory system that opens and closes banks on each reference, the latency associated with a memory reference is a page open and followed by CAS accesses. Typically the page open is somewhat slow (30 ns) and the CAS accesses are quick (10 ns each). Typically two or four CAS accesses are required, depending on the width of the memory system; we shall assume four in this report. After each access, the page is closed (30 ns), but while this is happening the data is already being returned to the processor, so it does not add to the latency.

In a memory system that leaves pages open, a 'page hit' (reference to the same row in a bank) takes only the CAS access time, since the page is already open. A 'page miss' is more expensive; the page that is currently opened must be closed, and then the new page opened, followed by the CAS accesses. In our previous report, we showed that proper selection of bank indexing bits makes it easy to obtain a high page hit rate, so this occurrence is infrequent (perhaps one in five accesses). The increased latency for the page misses is what you pay for the decreased latency for the page hits.

This report improves our previous results by showing how the effects of page mode interleaving and cache effect interleaving can be more efficiently combined when fewer memory banks are available.

One assumption from the previous report was that all banks are kept open always. This can lead to an overly large memory controller; each bank that is kept open has a certain amount of state and machinery associated with it. It is often desireable to keep track of fewer banks, even if the memory chips used have more actual banks.

There are a number of ways that a memory controller can have fewer bank controllers than the memory chips can support. First, banks can be combined in several ways: banks from multiple chips can be opened and closed together, thus increasing the size of the open pages. Alternatively, some memory system bank bits can be interpreted as row bits, increasing the number of rows per

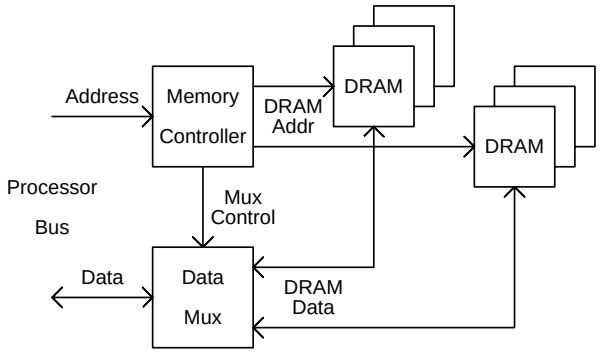

Figure 1: The memory system block design.

controller bank. Finally, the bank controllers can implement an associative tag arrangement so a smaller number of controllers can control a larger number of banks, but with only a subset of the banks open at a time. In this report, we show that this last alternative is the best.

In this paper, we consider the effect of only keeping some recently used pages open and closing the rest. Thus, in a system with 64 banks and 2 CPUs, we might keep only 8 banks open and the remaining 56 banks closed. Our hope is that keeping only 8 banks open will not reduce the page hit frequency significantly, but will allow most accesses that are not page hits to access banks that are already closed, so that they are not slowed down by open banks. Thus, the page miss rate (which is the cost of having a page-mode memory controller) is reduced.

2 Assumptions

We used the same traces, programs, and methodology as in the previous study, so the results are directly comparable. In addition, we assume that the results of that study are used in the design of the memory system, so that both page interleaving (to spread the memory demand among the banks of the system) and cache-effect interleaving (to separate writebacks and loads into different banks) are used in the bank indexing scheme.

3 Memory Controller Description

The memory system design we use is that depicted in Figure 1. There are three main components, the memory controller (MC), DRAM array, and data multiplexors/bus drivers (DMUX).

The memory controller is the 'smarts' of the system and it is responsible for all arbitration of busses, timing of commands, and page mode control. It contains state and state machines to control page-mode accesses and to make sure the timing requirements of the DIMMs and SIMMs are satisfied. It is connected to the address and control busses from the processor, and it drives the DMUX control lines. It generates DRAM commands for the DRAM array; if electrical loading or pin count is a concern, it may be necessary to introduce additional demultiplexors or simple state

machines between the memory controller and the DRAM array.

The DRAM array is some number of DRAM chips or SIMMs or DIMMs. One DRAM 'rank' is the number of chips over which a single cache line is distributed. Each rank may have multiple banks; DRAMs with 2 or 4 banks per chip are common, and it is expected that this number will rise in the future. Multiple ranks may share address and/or data lines, or they may be separated into separate busses to minimize contention and maximize memory-side bandwidth. It is important that the number of addressable banks is maximized, given the other constraints; the primary factor in improving page mode hit rate is the total number of addressable banks (and not necessarily the number of bank controllers in the memory controller or the number of banks open at a given time).

The data multiplexors/bus drivers (DMUX) chips are responsible for meeting the timing and loading requirements of the data bus. In almost all memory system designs, DMUX chips are required in order to connect a number of usually wide, slow, heavily loaded (several chips) DRAM busses to a faster, narrower bus with tighter loading requirements. To these DMUX chips we have added some number of prefetching and write buffers. The write buffers are needed to buffer data taken off the processor bus during a write command and hold it until the DRAMs can be made ready to accept the data. The prefetch buffers are used to hold data that was speculatively fetched from the DRAM array in hopes that it would be used. The control lines from the memory controller to the data multiplexors/bus drivers are used to indicate whether to drive or read from the DRAM and/or processor data busses, and whether to forward the data to the other bus or to store it to or read it from a particular buffer on the DMUX chips. For the remainder of this report we will ignore the write buffers and just consider prefetch buffers. While it is possible (and even advantageous) for these two functions to share the same pool of buffers, we shall simply assume that there is a fixed, finite number of prefetch buffers.

One thing to note about the data multiplexors and bus drivers is that they can be and are typically heavily bit sliced. Thus, for a 64-bit data bus and 256-bit DRAM bus, rather than have a single huge chip, this function is typically divided up into around eight smaller chips, each driving 8 bits of the data bus and 32 bits of the DRAM bus. This makes it much cheaper to support multiple DRAM busses (to reduce contention) and it also means that memory added to a single DMUX chip is multiplied by the number of DMUX chips in the system. If 16,384 bytes of static memory can be placed into a single DMUX chip, eight chips would have 256Kbytes of memory total, and the control would be no more complex than if there were just a single DMUX chip.

The memory controller has some number of internal bank controllers. At any time, each bank controller is either idle or else controlling a single bank. At any time, each bank is either idle (closed) or else controlled by a single bank controller. There can be more banks than bank controllers.

Each bank controller has some state. This state includes:

- State: one of INV (invalid), OPEN, or PREF (prefetched).
- Bank: the value of the bank index bits indicating which bank this controller currently controls; only valid if the state is OPEN or PREF.

- Page: the value of the row bits indicating which page in the bank is currently open; only valid if the state is OPEN or PREF.
- PrefetchLine: the value of the column bits indicating which cacheline is in what prefetch buffers if the state is Prefetched; if the state is Open, holds the address the most recent read request for this bank, plus one cacheline.
- PrefetchIndex: the number of the prefetch buffer that holds the prefetched data (if any).

The bank controllers are organized into an array. Indices to the bank controllers are maintained in a least-recently-used (LRU) stack. This stack supports two operations: move a particular bank controller index to the top of the stack, and return the index at a particular depth in the stack. This structure allows us to easily find the correct bank controller to 'reuse' by finding one sufficiently old; the general indexing (which is easily implemented) allows us to programmatically select the number of bank controllers that will be in use in a particular system.

The prefetch buffers in the data muxes are also organized into an array, and the indices into the prefetch buffers are maintained by the same LRU stack as that of the bank controllers. If the number of prefetch buffers is the same as the number of bank controllers, the machine is somewhat simplified and the same indices can be used for the prefetch buffers and the bank controllers.

The actions that cause state changes in a bank controller are as follows:

- From INV action open to OPEN: A read or write transaction from the bus to a closed bank will obtain a new bank controller by selecting the least recently used one; if this bank controller is in the INV state, the appropriate DRAM bank will be opened.
- From OPEN or PREF action open/close to OPEN: A read or write transaction from the bus to a closed bank will obtain a new bank controller by selecting the least recently used one; if this bank controller is in the OPEN or PREF state, and the bank being controlled is different from the bank corresponding to the read or write transaction, the OPEN of the new bank and the close of the old bank will proceed in parallel.
- From OPEN or PREF action close to INV: A read or write transaction from the bus to a closed bank will obtain a new bank controller by selecting the least recently used one; if this bank controller is in the OPEN or PREF state, and the bank being controlled is the same as the bank corresponding to the read or write transaction, the current page must be closed; a later open transition will open the new bank. This transition will also fire for all banks if the refresh timer has expired, indicating that it is time to refresh all banks.
- From OPEN or PREF action access to OPEN or PREF: A read or write transaction to the open page in an open bank but does not match in the prefetch buffer will simply perform a column access. If the access is a read access, the final destination will be OPEN and the PrefetchLine bits updated; if it is a write access, the final state is the same as the initial state and the PrefetchLine values are not modified. This transition also completes the read or write transaction.

- From OPEN action prefetch to PREF: If no transaction is outstanding for this bank, and the PrefetchLine value has been set, and the PrefetchIndex value is valid indicating that we have a prefetch buffer, a speculative prefetch is issued to fetch the line into the prefetch buffer.
- From OPEN or PREF action reclaim to OPEN: Whenever a bank controller falls to a depth in its LRU stack greater than the number of prefetch buffers, its current prefetch buffer is reclaimed and handed to the bank controller just pushed to the top of the LRU stack.
- From INV action refresh to INV: A refresh cycle is performed when all banks are in the idle state after the refresh timer has expired.

This is a somewhat simplified view. Most of the actions listed above take some amount of time to complete, so there may be potential state transitions from partially completed actions. Nonetheless the implementation should be relatively straightforward.

From the perspective of a memory transaction, what happens is as follows.

- A bank controller is selected. The bank index bits of the memory reference is compared against the Bank bits of all the memory bank controllers; if one matches, this is the bank controller for that reference. Otherwise, the least recently used bank controller is the bank controller for that reference. In any case, this bank controller is moved to the top of the LRU stack, and is given a PrefetchIndex to a prefetch buffer if it does not already have one (which may initiate a reclaim in some other bank controller).
- The prefetch status is checked. If the Bank, Page, and PrefetchLine bits all match the address of the reference, and the reference is a read, and the state of the bank controller is PREF, then the data is driven directly to the bus to complete the transaction. The PrefetchLine bits are set to the value from this reference and the state of the bank controller is changed to OPEN to initiate the next prefetch. The remaining steps in this list are skipped. If the reference was a write and the bank controller state is PREF, a number of policies can be implemented. If the write is to the same line as the current PrefetchLine, the state of the bank controller can simply be changed to OPEN. Or, whether or not the write is to the same line, the state can be changed to OPEN. Because the cache effect interleaving of the bank indexing scheme tends to separate writes and reads to separate banks, there should be little penalty for simply always changing the state to OPEN on a write.
- The page is opened. If the state of the bank controller is not INV and both the Bank and Page bits match those of the reference address, the page is already open and this step is skipped. Otherwise, if the state of the bank controller is not INV and the Bank bits match but the Page bits do not, then the wrong page on the bank is open, and we have to close the page and reopen the correct page serially. This we call a page miss. If the Bank bits are different or the state of the bank controller is INV, then we can close the old bank (if any) and open the new bank in parallel; we call this an idle bank reference. After opening the page, the new state is OPEN.
- The reference is performed through an access action. If the transaction is a read, the Prefetch-Line bits are set from the reference address the resulting state is OPEN (from which a new

prefetch might be initiated). Otherwise, the resulting state is the same as the state at the beginning of this step. In any case, this access transition completes the transaction and the bank is left open.

References may be classifed into sequential hits, page hits, idle bank references, and page misses. A sequential hit is a reference for which the data was already in a prefetch buffer or was in the process of being transferred into a prefetch buffer. The total latency from address on processor bus to critical word returned on data bus for such a reference might be 30 ns. A page hit is a reference for which the correct page in the correct bank was already open, and it includes all sequential hits. The latency for this form of reference will be higher—perhaps 90 ns—because the DRAMs still need to be queried. A page miss is a reference for which some incorrect page was open in the requested bank, and thus the page close and new page open had to be performed in sequence. Such a reference might take 150 ns. An idle bank reference is a reference to a bank in which no page was open; this form of reference might take 120 ns. A memory controller that does not support page mode or prefetching will see a memory latency of 120 ns for each reference, neglecting contention. We shall use these values as typical throughout the remainder of this report when presenting performance figures.

Initiating a prefetch when no access is pending has very little effect on the latency of a following access because of the CAS latency and pipelining in the SDRAM. If no read is initiated in a particular cycle, that cycle is lost forever, and for most SDRAMs, initiating a read in a particular cycle delays a subsequent read by at most 0 or 1 cycle. There may be delays due to contention for the DRAM address or data busses, or for DRAM data bus turnaround time, but we expect that the extra bandwidth normally available on these busses in a competitive memory design will make these effects negligible.

Prefetching cache lines into the data multiplexor also can lead to significant latency savings. First, all latency due to the DRAM is eliminated on a prefetch hit (if the data is already in the buffers). This includes the CAS latency, but it also includes what may be a more significant contributor to the latency: the time required to drive the DRAM address and DRAM data pins. Both of these busses generally suffer from significant fan-out and fan-in, and this leads to additional latency. A prefetch hit, on the other hand, involves a critical path that includes only the memory controller chip, multiplexor control signals, and data multiplexor chips.

4 The Xor Bank Indexing Scheme

Our previous paper [TR96] considered what address bits should be used to select memory banks in order to optimize the page hit rate and sequential prefetch hit rate. An unstated assumption of that paper was that the address bits were used directly; that is, the mapping from the address to the bank selection was just the identification of a subset of the address bits. That paper identified two phenomena, page interleaving and cache effect interleaving, and associated a set of the bits from the address with each effect. The optimal bank indexing scheme simply distributed the bank selection bits among these two effects, as shown in Figure 2. The optimal such bit patterns are shown in Figure 3.

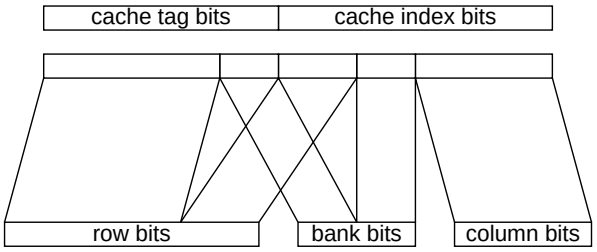

Figure 2: The old bank indexing technique, showing how the address bits are permuted into row bits, bank bits, and column bits. The top box shows the division of the cacheline address into cache tag bits and cache index bits. The middle box represents the actual cacheline address, from most significant bits to least significant bits. The bottom boxes are the individual components for the memory system.

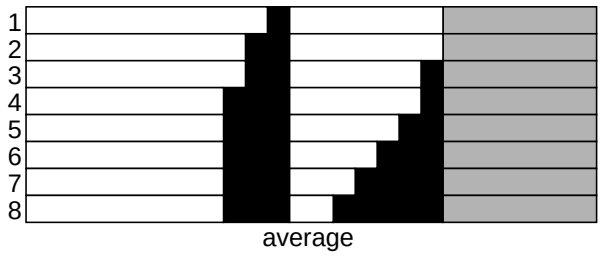

Figure 3: The best bit patterns on average. Gray indicates the bit was used for column addressing, black indicates the bit was used for bank indexing, and white means the bit was used for row addressing. The number on the left is the total number of bank bits. The leftmost set of bank indexing bits is coincident with the lowest tag bits for a 4 megabyte, 4-way set associative cache and thus implement cache-effect interleaving; the rightmost set of bank indexing bits implements page-effect interleaving. These bank indexing schemes are those that had the highest average page hit percentage over all five traces.

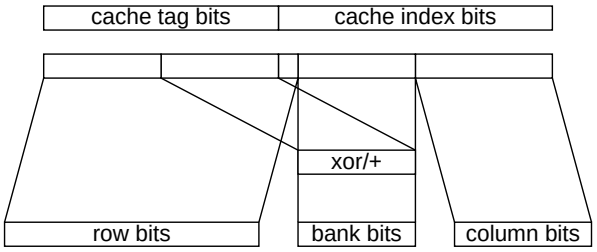

Figure 4: The xor bank indexing technique, showing how the bank bits are derived from the address using either the exclusive or operation or the addition operation.

With a tiny bit of logic, we can improve on these bank indexing schemes. Rather than simply selecting address bits, we can combine bits from the cache effect region (the least significant address bits coincident with the cache tag bits) with bits from the page effect region (the least significant address bits more significant than the column address bits) using an exclusive or, addition, or other operation, as shown in Figure 4, to compute the bank index bits.

The main purpose of cache effect interleaving is to separate the replacement writebacks and the read misses into separate banks so they do not thrash a single bank. Since the read miss and the line it replaces have the same cache index bits, the page effect interleaving bits are the same. Thus, using addition or exclusive-or will preserve this separation.

The purpose of page effect interleaving, on the other hand, is to spread the memory usage across as many banks as possible. Combining the cache tag bits with the page effect interleaving region will tend to slightly improve the distribution of the addresses across the memory banks.

To illustrate how exclusive-or bank indexing improves the page hit rate, consider the case where we only have eight memory banks, and thus only three bank select bits. The 'optimal' indexing scheme found in our prior paper allocated two of those bits to the cache effect region and one of those bits to the page region, as shown in Figure 3. With only two bits in the cache effect region, there is about a 1 in 4 chance that a miss and the cacheline being replaced have the same value for these bits, so this leads to a certain amount of bank thrashing. Similarly, with only a single bank index bit in the page interleaving address region, alternating memory pages will almost always belong to the same bank, so hot regions of memory larger than two DRAM memory pages will tend to thrash.

With exclusive or bank indexing, using three bits, we combine three bits from the cache effect interleaving bits with three bits from the page interleaving bits using the exclusive or operation to yield our three bank index bits. Now, a miss and the line being replaced have only a 1 in 8 chance of belonging to the same bank index region. Also, sequential memory pages are spread across all eight banks, so large hot regions of memory are maximally spread. With this change, the page hit rate improves from 63.3% to 68.2%.

In addition, writes tend not to be latency sensitive, and reads have a higher page hit rate than

do writes. The previous report reported the overall page hit rate, which is important for bank throughput, but less important for latency. When we only focus on reads, the actual hit rate for the old bank indexing scheme rises from 63.3% to 68.1%, and for the exclusive-or bank indexing scheme from 68.2% to 72.7%.

The improvement is shown in Table 1, which compares the results for the old optimal bank indexing scheme with those resulting from xor indexing. The improvement is most dramatic for four to sixteen banks. The sequential prefetch hit rate also improves significantly.

	old			xor		
Bank bits	hit	read hit	prefetch	hit	read hit	prefetch
0	0.250	0.319	0.250	0.250	0.319	0.250
1	0.403	0.458	0.334	0.413	0.478	0.346
2	0.513	0.557	0.396	0.569	0.623	0.444
3	0.633	0.681	0.489	0.682	0.727	0.511
4	0.718	0.752	0.533	0.754	0.796	0.556
5	0.765	0.799	0.556	0.795	0.836	0.574
6	0.800	0.833	0.569	0.829	0.869	0.584
7	0.834	0.866	0.581	0.852	0.891	0.589
8	0.866	0.896	0.584	0.891	0.921	0.595

Table 1: The attainable page hit rate, read hit rate, and sequential prefetch hit rate using the old 'optimal' bank indexing scheme and the improved exclusive-or bank indexing scheme.

There are some restrictions on the function that can be used for combining the bank indexing. Specifically, the mapping from the cacheline address to the set of row, column, and bank bits should be 1:1. Both addition (ignoring any carry out) and exclusive-or satisfy this requirement; multiplication and logical or are two functions that do not satisfy this requirement.

It is possible and not harmful for the page interleaving bits to crowd into the cacheline interleaving bits if a large number of bank bits is used. Indeed, perhaps the simplest implementation of this technique is to just add or exclusive-or a full eight bits from the cacheline interleaving region to eight bits from the page interleaving region. Then, select an appropriate number of bits from the result to be used to select which bank, and use however many high-address bits are required for row selection. Thus, the implementation of this technique might be simpler than that for the old bank indexing scheme.

If cacheline (or other small) interleaving is required for whatever reason, such as when the SDRAM data bus residency is higher than the CPU bus residency, the same technique applies; some of the cacheline effect bits should be added or exclusive-or'ed with the cacheline interleave bits. This will tend to put writebacks in a different bank than the misses. In this case, careful consideration should be given to the use of exclusive-or or addition. Addition will preserve the sequential nature of accesses, so that bursts of sequential accesses will always use the busses in the same order; the exclusive-or function will tend to permute the bus order of bursts of sequential accesses. With

addition (or without any special bank indexing scheme), processors will tend to synchronize their sequential bursts with other processors also doing sequential bursts. If each processor is running the same code, issuing the misses at roughly the same rate, this can be beneficial. If the processors are running different code, so that the sequential accesses would have different delays between them if run independently, this synchronization can slow down all processors to the speed of the slowest processor. In this case, the 'randomization' effect of exclusive-or might be beneficial. Other factors must be taken into consideration, such as the number of busses and total overbandwidth. With significant overbandwidth, the exclusive-or technique will introduce very little blocking; with only slight overbandwidth, the exclusive-or technique will introduce more blocking, so the additional technique may be appropriate. The evaluation of cacheline interleaved systems is beyond the scope of this report, and it is not even clear that such effects occur with any significant frequency in the real world, but there is great potential for non-obvious and degenerate behavior that must be carefully considered in such a system. Potential solutions may include sufficient overbandwidth to reduce the frequency of such behavior and intentional delays or perturbation in the order requests are satisfied.

5 Experiments

We used our improved bank indexing scheme to investigate the idea of only keeping a subset of the memory banks open, in order to minimize the page miss penalty. The remainder of this report focusses on this idea.

The bank indexing scheme simulator was modified to support an LRU stack for each bank. Each time a bank was referenced, that bank was moved to the front of the LRU stack. Each time a page hit, page miss, or sequential hit occurred, the location on the LRU stack was reported. This allowed a single simulation to compute results for an LRU stack of any size, in a way similar to that used for running multiple cache configurations in a single run.

The experiments did not model contention, or consider whether there was enough time to actually complete a prefetch. The experiments followed the simple memory controller description and state diagram given above.

6 Results and Analysis

The primary question of interest is, as we reduce the number of bank controllers below the number of memory banks, what is the impact on the sequential prefetch rate, the page hit rate, and the page miss rate?

Figure 5 shows the primary results for a sample memory system with 32 banks. (Such a system might be composed of DRAMs with 4 banks each chip, and eight separate DRAM busses; a multiprocessor system might use such a configuration. Alternatively, it might be composed of eight ranks of DRAMs sharing a single bus; this might be more typical in a uniprocessor machine.) The 'kept' variable is the fraction of banks that are kept open; when multiplied by the 32 banks, this is the number of bank controllers we need (and thus, to some extent, the size and complexity of our memory controller chip). In this and all other graphs in this report, the reported numbers

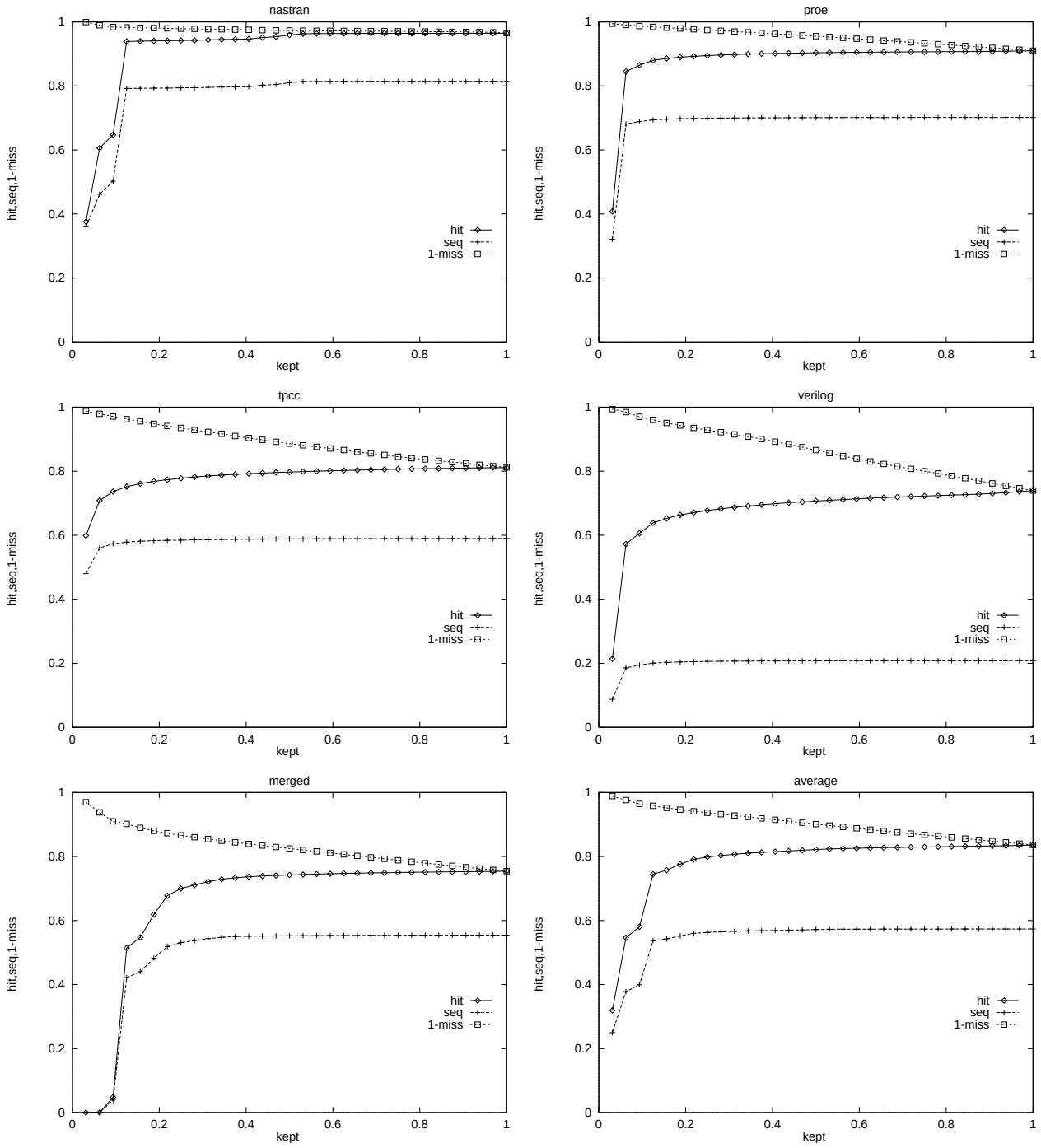

Figure 5: The impact of maintaining fewer banks open than the memory supports. These charts are for a memory system with a total of 32 banks; the three lines in each graph represet the sequential hit rate, the page hit rate, and the number of references that are not page misses. These graphs represent the xor bank indexing scheme. The x axis is the fraction of banks kept open; as it reaches one, the number of page hits becomes equal to the number of references that are not misses since there are no idle banks.

represent only the read transactions; while the write transactions were simulated and taken into account, their latency is much less important so we omit them from our graphs. These graphs make it evident that the very first few banks kept open yield almost all of the hit rate and sequential prefetches, and with only a few banks kept open, most references that are not page hits are idle bank references rather than page misses. Indeed, as the number of banks kept open increases past about 8 for this configuration, typically the incremental change in the page hit rate is less than the incremental change in the page miss rate. Table 2 summarizes the page hit, page miss, and sequential prefetch rates for different numbers of bank controllers, and lists the average memory latency for such a system. Only eight controllers are required to attain the minimum average mem-

Controllers	Seq	Hit	Miss	Latency
1	0.250	0.319	0.011	96
2	0.378	0.546	0.024	82
4	0.537	0.745	0.042	67
8	0.563	0.798	0.063	64
16	0.572	0.822	0.099	64
32	0.574	0.836	0.164	65

Table 2: The attainable page hit rate, page miss rate, and sequential prefetch hit rate for different numbers of bank controllers. This table assumes the memory system supports 32 banks.

ory latency of 64 ns. Of the 56 ns saved from the normal 120 ns memory latency, 22 ns comes from the page mode support and an additional 34 ns comes from sequential prefetching.

The optimal performance point depends on the relative costs of a page closing and a page opening, but in general, the change in performance from using only eight bank controllers versus 32 bank controllers is probably negligible, especially since all of these traces had such a high page hit rate.

It is instructive to plot how the hit rate and miss rate depend on each other, as illustrated in Figure 6. This graph also presents all of the results for the number of memory banks from 2 to 256. On these graphs, the lower right hand corner is the highest performance; the upper left hand corner is the lowest performance. Each line on the graphs represents a constant number of total memory banks. The number of bank controllers increases from left to right with the hit rate; all values from 1 to the number of memory banks were run and are marked on the graph. These graphs make it even more evident that the first few (half dozen) bank controllers increase the hit rate quickly with only a modest increase in the miss rate; the curves are mostly horizontal in this region. As the number of controllers approaches the number of banks, the curve turns upward and starts increasing the miss rate along with the hit rate. Especially for nastran, and with eight or more memory banks, these applications tend to achieve a very good hit rate, so their miss rate never gets very large.

Another view of the same data is presented in Figure 7 and summarized in Table 3. In this figure we hold the number of bank controllers fixed at eight, as we vary the number of memory system banks from eight to 256 by powers of two. In general, our page hit rates and sequential hit rates improve. The number of references that do not miss also increases. Thus, a simple memory controller with

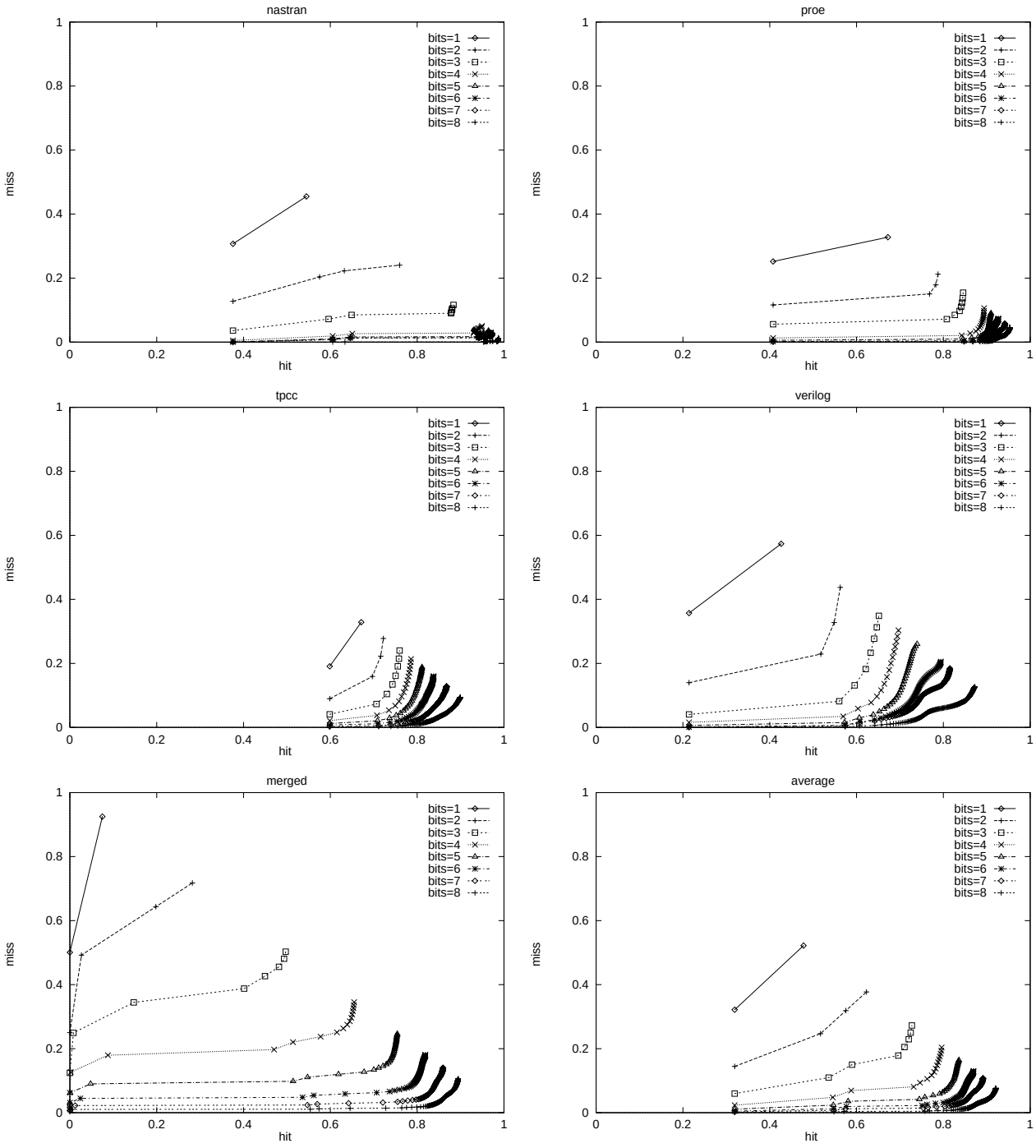

Figure 6: The relationship between the page hit rate and the page miss rate as the number of banks kept open is increased for different number of memory system banks. The ends of the lines coincide with the line miss + hit = 1.

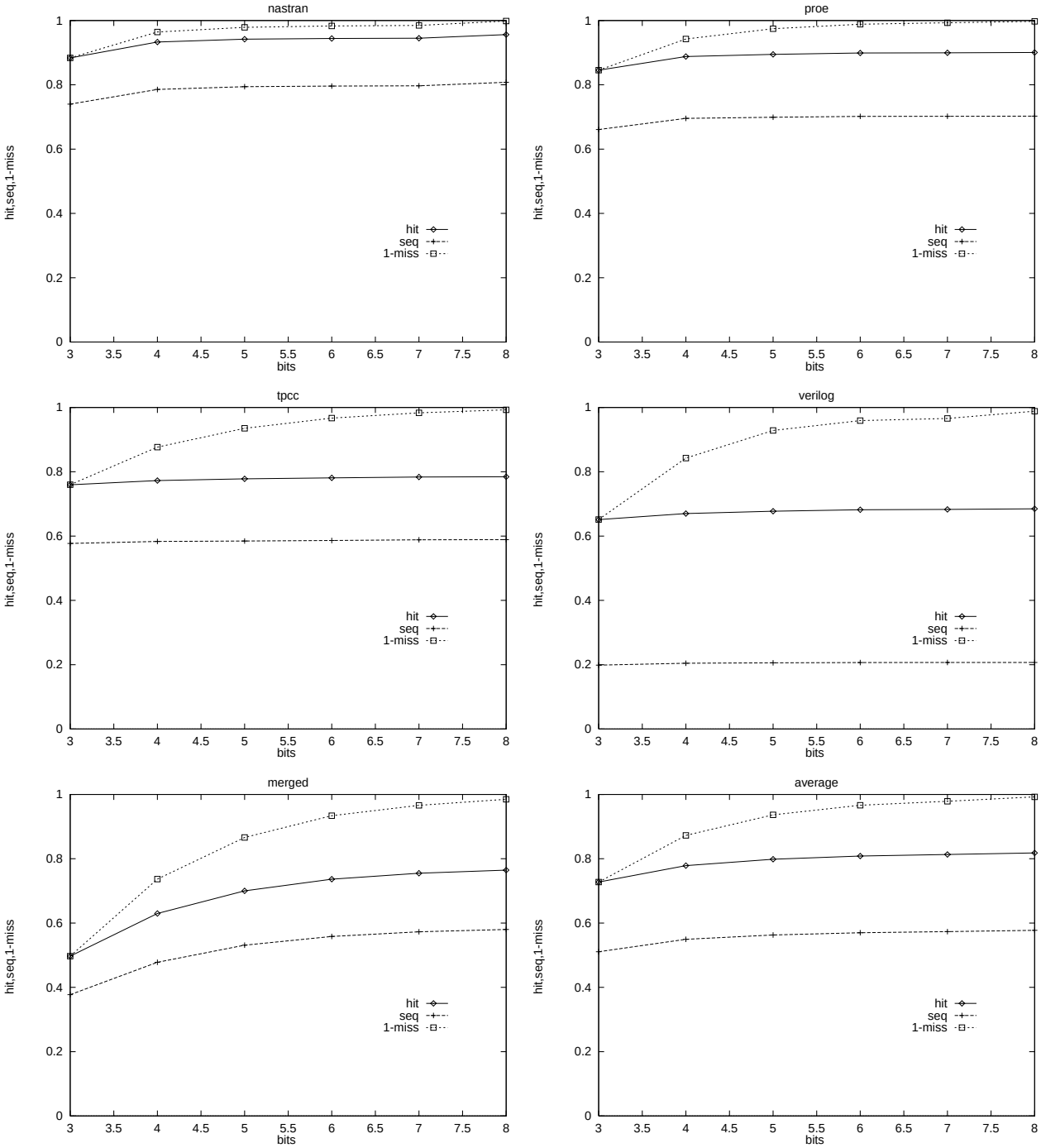

Figure 7: The impact of increasing the number of memory system banks while holding the number of bank controllers fixed (in this case, we hold 8 banks open at a given time).

only eight internal bank controllers works better and better as the number of banks in the memory system increases.

Banks	Seq	Hit	Miss	Latency
8	0.511	0.727	0.272	76
16	0.549	0.779	0.127	67
32	0.563	0.798	0.063	64
64	0.570	0.808	0.034	63
128	0.573	0.813	0.021	62
256	0.577	0.818	0.008	61

Table 3: The attainable page hit rate, page miss rate, and sequential prefetch hit rate for different numbers of memory banks, holding the number of bank controllers fixed at 8.

7 Alternative Bank Organizations

When the number of bank controllers is smaller than the potential number of banks, there are several ways to organize the resulting memory system other than the design presented in this paper. The design in this paper leaves the 'extra' bank bits as bank bits, and uses an associative bank controller design to dynamically map a limited number of controllers over a larger set of memory banks, gaining advantages in an increased hit rate, increased sequential prefetch rate, and decreased page miss rate.

One alternative is to essentially treat 'extra' bank bits as row bits. This is the simplest alternative, and like the design in this paper, it often allows a new page to be opened in parallel with the closing of an old one (resulting in a decreased page miss rate). But the page hit rate and sequential prefetch rate are identical to that of a memory system that had a number of banks equal to the number of real bank controllers. We reran our simulations illustrated in Figure 7 for this case, and the result is shown in Figure 8; this is the case with eight bank controllers and a memory system supporting between 8 and 256 banks. As we can see, in this case, additional memory system banks do not increase the page hit rate or the sequential prefetch hit rate, although the miss rates do improve. For all three metrics, using the memory design presented in this paper is superior. Table 4 summarizes the data in these graphs.

Another alternative memory system design is to control multiple banks in parallel with a single memory bank controller, effectively increasing the page size. This alternative is essentially using extra bank bits as column bits. The results for this alternative are presented in Figure 9 and summarized in Table 5. In this case, the page miss rate is always equal to one minus the page hit rate, since all banks are open at all times, so the page miss rate line is not shown in the graphs. The page hit rate and sequential page prefetch rate lines do somewhat interesting things. In particular, the page hit rate increases more rapidly for some applications (tpcc and verilog) than it does for the first design presented in this paper. On average, the page hit rate stays about the same as for the first design. Secondly, the sequential prefetch rate actually falls for some applications. The

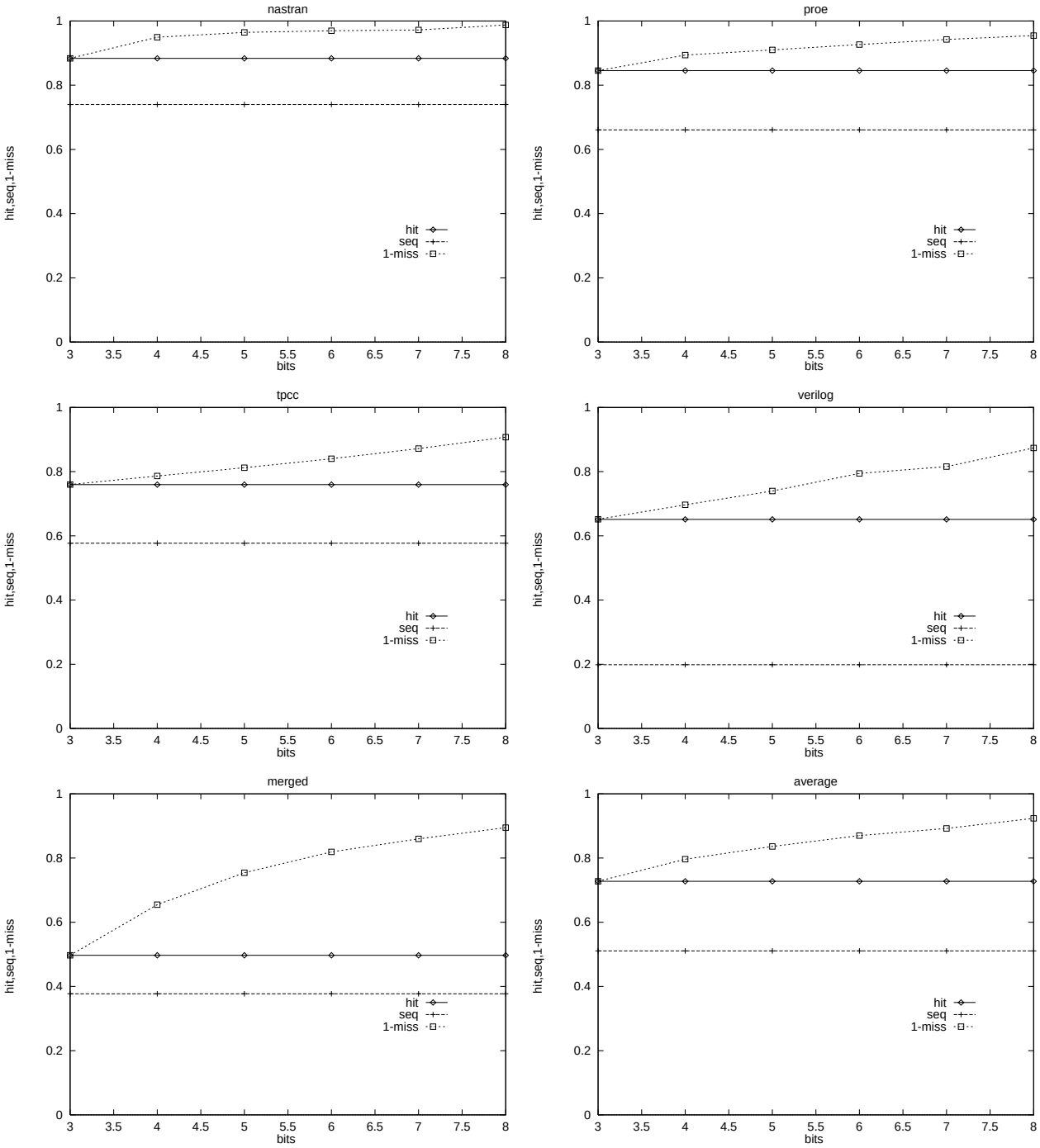

Figure 8: The impact of increasing the number of memory system banks while holding the number of bank controllers fixed using extra bank bits as row bits.

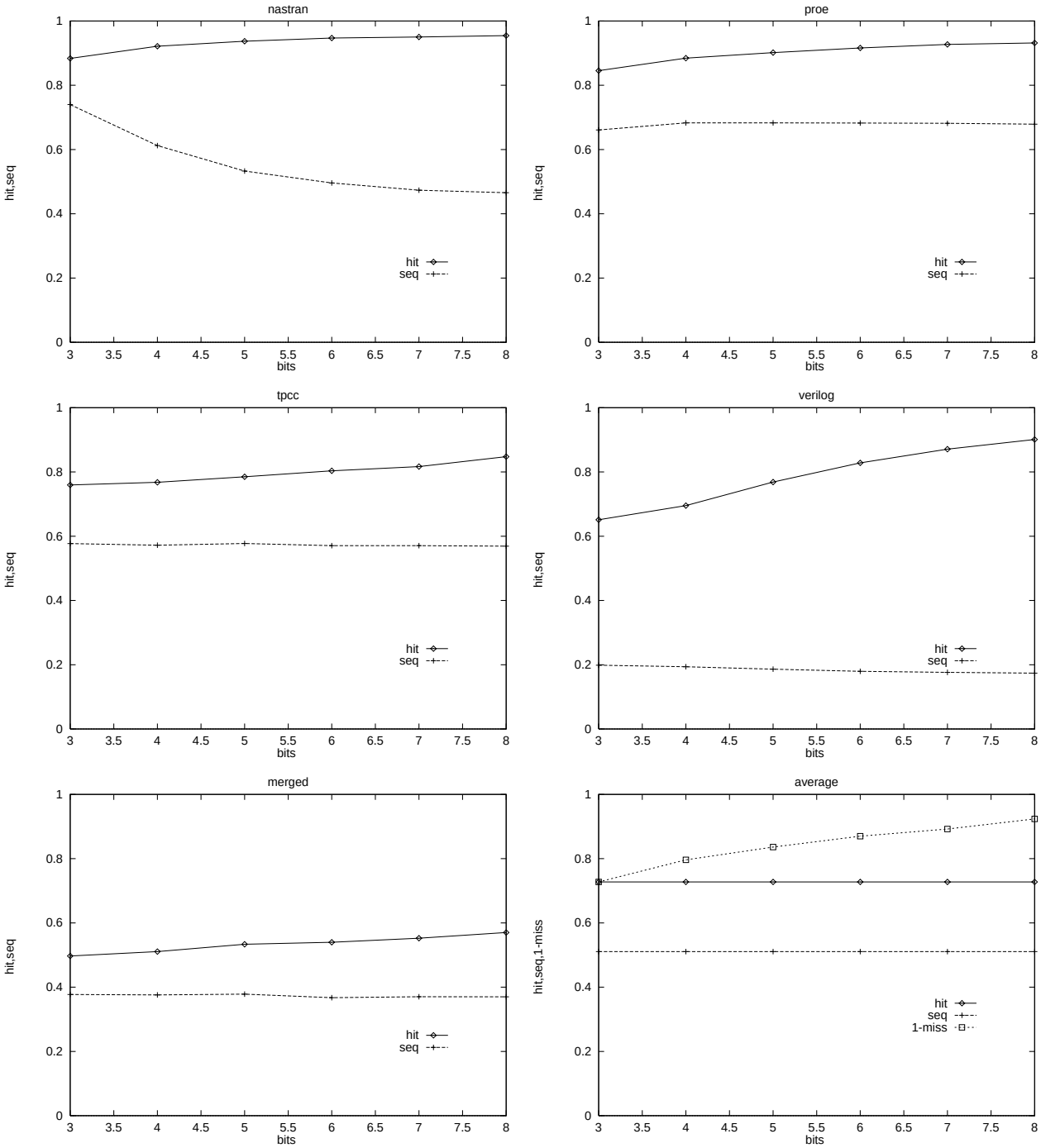

Figure 9: The impact of increasing the number of memory system banks while holding the number of bank controllers fixed using extra bank bits as column bits; see text for warnings.

Banks	Seq	Hit	Miss	Latency
8	0.511	0.727	0.272	76
16	0.511	0.727	0.204	74
32	0.511	0.727	0.164	72
64	0.511	0.727	0.130	71
128	0.511	0.727	0.108	71
256	0.511	0.727	0.076	70

Table 4: The attainable page hit rate, page miss rate, and sequential prefetch hit rate for different numbers of memory banks using extra bank bits as row bits.

Banks	Seq	Hit	Miss	Latency
8	0.511	0.727	0.273	76
16	0.487	0.756	0.244	75
32	0.472	0.785	0.215	75
64	0.459	0.807	0.193	74
128	0.454	0.823	0.177	73
256	0.451	0.841	0.159	72

Table 5: The attainable page hit rate, page miss rate, and sequential prefetch hit rate for different numbers of memory banks using extra bank bits as column bits.

primary reason for these results is that the resulting memory page size is larger (ranging from 8K to 256K), and this does help. Unfortunately, since these page sizes are larger than most operating systems' physical page size, and for nastran we are using a virtual trace, it is unclear how much of this spatial locality will survive the virtual to physical translation. In addition, the sequential prefetch rate was hurt by the larger pages since the sharing of a single prefetch line over a larger memory region meant fewer prefetch hits. A prefetch cache rather than bank-oriented prediction lines can help this, as we show in the next section. On average, however, the page hit rate declined, and the sequential prefetch hit rate declined significantly, and the page miss rate was substantially higher than for the primary design presented in this paper.

8 Using a Prefetch Cache

One problem with using a prefetching scheme based on banks, with only a single prefetch line per bank, is that a page hit will often cause a prefetch that replaces a previously prefetched line from that page, and that previously prefetched line is subsequently requested. A different prefetch storage alternative is to use a prefetch cache; prefetched lines are stored into a conventional cache. Table 6 compares the prefetching hit rates for a memory system with 256 banks and both 8 and 256 bank controllers with one prefetch line per bank controller with that of a memory system with a 512 byte, 8-way and 16 kilobyte, 16-way set associative prefetch cache. The 8 bank controller case uses the same amount of memory as the 512 byte prefetch cache case, and the 256 bank controller case

App	8/256 Bank Seq	256/256 Bank Seq	512 byte Pref Cache	16Kbyte Pref Cache
nastran	0.808	0.834	0.923	0.958
proe	0.703	0.708	0.689	0.709
tpcc	0.589	0.602	0.613	0.650
verilog	0.207	0.212	0.216	0.249
merged	0.580	0.621	0.527	0.624
average	0.577	0.595	0.594	0.638

Table 6: The attainable sequential hit rate when using a prefetch cache to hold the sequential prefetches rather than just a line per bank. We used 256 banks and thus 256 lines for the bank case, and both a 512 byte and a 16 kilobyte 16 way cache for the prefetch cache case.

uses the same amount of memory as the 16 kilobyte prefetch cache case. The prefetch cache makes better use of the available storage, and this advantage increases as the number of memory banks decreases, since with a prefetch cache the prefetch hit rate is totally independent of the number of memory banks in the system. Separating the prefetching cache from the memory bank controller may simplify the overall memory controller.

9 A Tough Workload

All of the traces we have discussed so far have a relatively high page hit rate and sequential prefetch hit rate. One of the features of the memory design presented in this report, however, is that in the presence of a bad page hit rate, the penalty for using a page mode memory controller is reduced. In order to present results supporting this, we searched for an application with a poor page hit rate. We finally found one—an eight-CPU, 64 process multiprocessor run of the TPC-C benchmark. In order to get the page miss rate high, we assumed a very fast CPU and a slow memory system such that the CPU could execute an average of 300 instructions during one memory reference. This might correspond to a CPU capable of executing two billion instructions per second and an average memory system latency of 150 ns. Such a fast CPU would tend to saturate the memory system; all the CPUs would be aggressively accessing memory and thrashing banks. (A more realistic CPU model would have bursty misses and somewhat long pauses between the bursts during which the CPU was executing out of cache; such a workload would have a higher page hit rate because fewer CPUs would be accessing memory simultaneously, on average.) This trace is significantly longer and has a larger working set than the tpcc trace we used earlier in this report. Using a large number of processors and processes causes significant interference in the caches and memory pages of the system. With a normal page-mode memory controller, the large fraction of page misses would cause most references to see a higher latency.

A memory system designed for a large SMP box should contain more bandwidth and more independent memory banks than a memory system designed for a uniprocessor small SMP box, simply because the memory demands tend to be greater and the potential for contention is higher. Thus, the results for this trace are not directly comparable with those of the uniprocessor traces; it makes no sense to build an 8-way multiprocessor with fewer than 8 banks of memory, and 16 or 32 banks

will certainly yield more reasonable performance.

The performance for this workload is shown in Figure 10 and summarized in Table 7. Even for this tough workload, it is possible to reduce memory latency by 26% using 256 banks and only 16 bank controllers. In this case, 43% of the memory references are not page hits, but only 3.6% incurred a page close penalty. In all cases, memory latency is less than that for a memory controller that does not implement page mode.

Memory Banks	Bank Controllers	Seq	Hit	Miss	Latency
16	4	0.096	0.216	0.219	114
16	8	0.181	0.410	0.352	107
16	16	0.190	0.451	0.549	112
32	8	0.200	0.450	0.180	100
32	16	0.218	0.516	0.285	100
32	32	0.223	0.547	0.453	104
64	8	0.210	0.470	0.092	96
64	16	0.233	0.548	0.147	94
64	32	0.241	0.593	0.239	95
64	64	0.244	0.619	0.381	98
128	8	0.215	0.480	0.046	94
128	16	0.240	0.565	0.074	91
128	32	0.250	0.617	0.121	90
128	64	0.255	0.657	0.200	91
128	128	0.257	0.677	0.323	94
256	8	0.217	0.486	0.022	93
256	16	0.244	0.574	0.036	89
256	32	0.254	0.630	0.059	88
256	64	0.260	0.677	0.099	87
256	128	0.264	0.710	0.167	88
256	256	0.265	0.726	0.273	90

Table 7: Some performance figures for the extreme TPCC workload.

The key thing to note about the curves in Figure 10 is their shape. The first two curves show that the page hit rate goes up rapidly initially, and then levels off, while the page miss rate tends to fall much more steadily. Thus, there is a point at which we can operate our memory controller such that we catch the great majority of the page hits, but most references that are not page hits are just idle references rather than page misses. This point is usually with a number of bank controllers at around two to four times the number of CPUs. Also, the last three graphs show that holding the number of bank controllers fixed but increasing the number of memory banks primarily reduces the number of page misses, while increasing the number of page hits and sequential prefetches gradually. Thus, a small number of bank controllers can easily utilize as many banks as the DRAMs support.

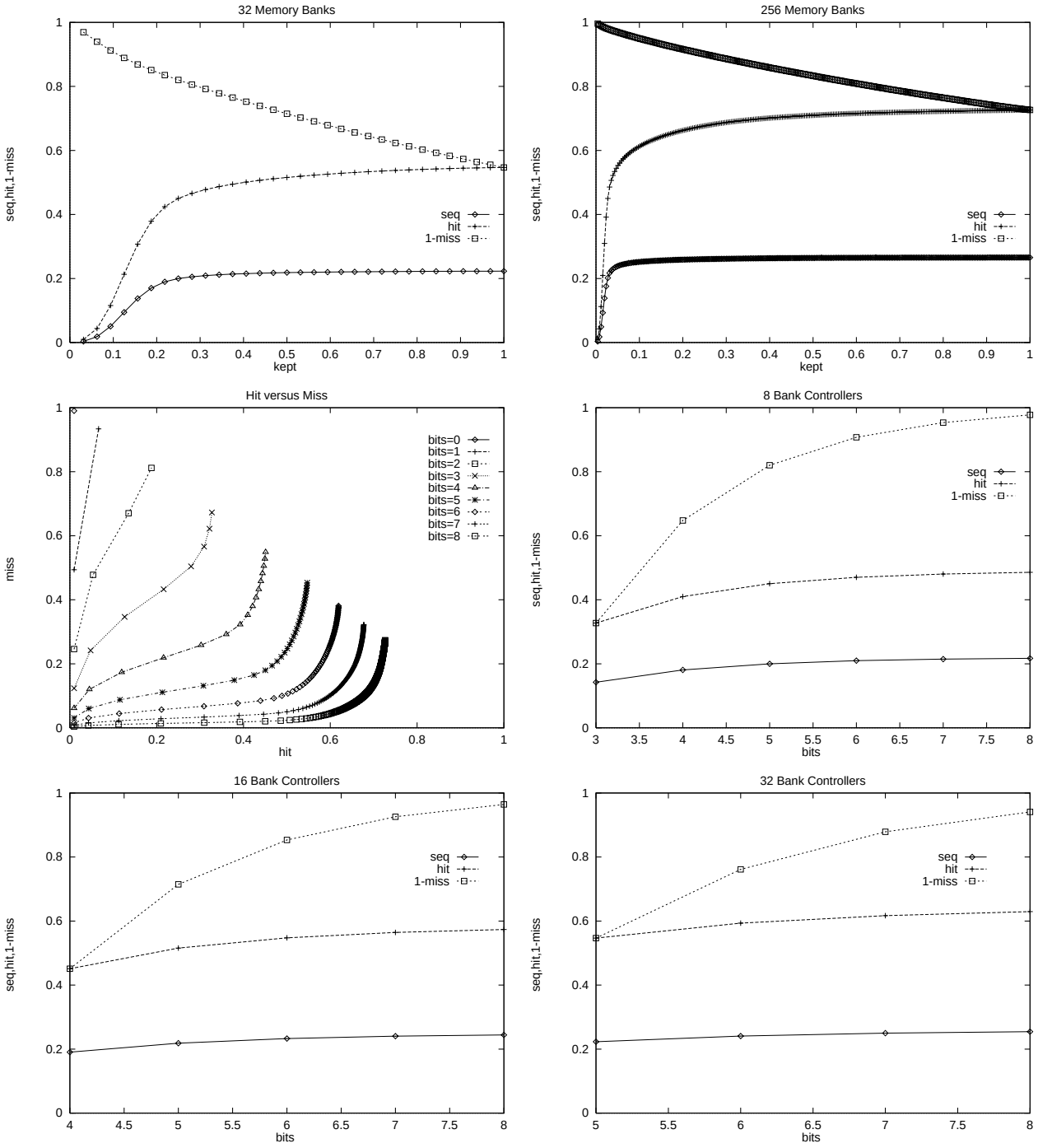

Figure 10: The performance for the extreme multiprocessor TPCC workload. The top two charts show the impact of the number of bank controllers as the number of memory banks is held fixed for 32 and 128 memory banks. The middle left chart shows how the hit rate and miss rate are related for various constant numbers of memory banks as the number of bank controllers is varied. The last three charts show the performance of the memory controller with 8, 16, and 32 bank controllers for numbers of memory banks.

10 The Impact of Refresh

DRAMs need to be refreshed in order to retain their content. A typical DRAM needs to be refreshed at a rate of one row every 16 microseconds. Refreshing a row closes all banks in current DRAM architectures. We expect the impact of this on the page mode architecture to be small, but in order to verify this, we modified our simulator to close all banks every n references, where n was a parameter supplied by the user. If references arrive about every 100 ns, then $n = 160$, so this is the value we used. For a multiprocessor, this value might be higher. The lower the value, the more impact refresh might have; at the extreme, with only one reference per refresh, we can never take advantage of page mode (unless we reopen banks after refresh, but that is probably too much complexity for very little gain).

The action of refresh is indeed much like the action of limiting the number of open banks at a given time. If we have a total of 32 memory banks, Table 8 shows the effect of refresh on the prefetch hit rate, page hit rate, miss rate, and memory latency, for various numbers of bank controllers. The

Controllers	Refresh?	Seq	Hit	Miss	Latency
1	N	0.250	0.319	0.011	96
1	Y	0.248	0.317	0.011	96
2	N	0.378	0.546	0.024	82
2	Y	0.375	0.541	0.023	82
4	N	0.537	0.745	0.042	67
4	Y	0.530	0.734	0.041	67
8	N	0.563	0.798	0.063	64
8	Y	0.554	0.785	0.060	65
16	N	0.572	0.822	0.099	64
16	Y	0.562	0.803	0.086	65
32	N	0.574	0.836	0.164	65
32	Y	0.562	0.808	0.106	65

Table 8: The attainable page hit rate, page miss rate, and sequential prefetch hit rate for different numbers of bank controllers, both with and without refresh. This table assumes the memory system supports 32 banks.

impact is small for all of our simulations. The page hit rate and sequential hit rate is reduced a few percent, but the page miss rate tends to be also reduced by the same or a greater amount.

11 The Impact of Cache Line Size

Both page mode hit rate and sequential hit rate take advantage of spatial locality in the reference stream. A larger cache line size also takes advantage of spatial locality. With a page-mode memory controller, as the cache line size increases, the amount of spatial locality left in the reference stream that is seen by the memory system decreases. In this section, we evaluate this effect.

Table 9 shows how the traffic changes as the cache line size changes. Nastran, with its extremely

high degree of spatial locality, has a nearly constant total number of bytes transferred, independent of the cache line size. Verilog, on the other hand, has a large increase in total byte traffic as the cache line size increases. The other applications lie somewhere in between.

	Total Transactions			Total Bytes		
App	32	64	128	32	64	128
nastran	1.988	1	0.503	0.994	1	1.006
proe	1.832	1	0.557	0.916	1	1.115
tpcc	1.720	1	0.594	0.860	1	1.188
verilog	1.372	1	0.746	0.686	1	1.492

Table 9: The change in traffic, both transactions and total bytes, as the cache line size increases. All values are normalized to the case of a cache line size of 64 bytes. Note how nastran has a nearly constant total number of bytes, while verilog has a significant increase in total bytes as the cache line size increases.

Figure 11, summarized in Table 10, shows the impact of the cache line size on the page mode hit rate and the sequential prefetch hit rate. For nastran, the amount of spatial locality is so high that increasing the cache line size has virtually no impact on the sequential prefetch hit rate. The other applications show a significant drop in the page mode hit rate as the cache line size increases, but even at a cache line size of 128 bytes, the page mode hit rate is still fairly high.

	Page Hit Rate			Sequential Prefetch		
Banks	32	64	128	32	64	128
1	0.366	0.321	0.280	0.278	0.251	0.220
2	0.526	0.479	0.430	0.376	0.347	0.307
4	0.676	0.624	0.564	0.476	0.445	0.396
8	0.778	0.728	0.665	0.540	0.511	0.456
16	0.844	0.796	0.735	0.583	0.556	0.499
32	0.879	0.836	0.779	0.599	0.574	0.517
64	0.906	0.870	0.819	0.607	0.584	0.527
128	0.924	0.891	0.845	0.612	0.589	0.534
256	0.947	0.921	0.883	0.618	0.595	0.541

Table 10: The change the page mode hit rate and sequential prefetch hit rate as the cache line size changes for different numbers of banks.

12 Discussion

We have presented a memory system design that, with only minor modifications, can cut the average memory latency in half. In addition to using page mode optimally (as presented in the previous report), we use two additional ideas to reduce latency. First, we keep only a fraction of the total

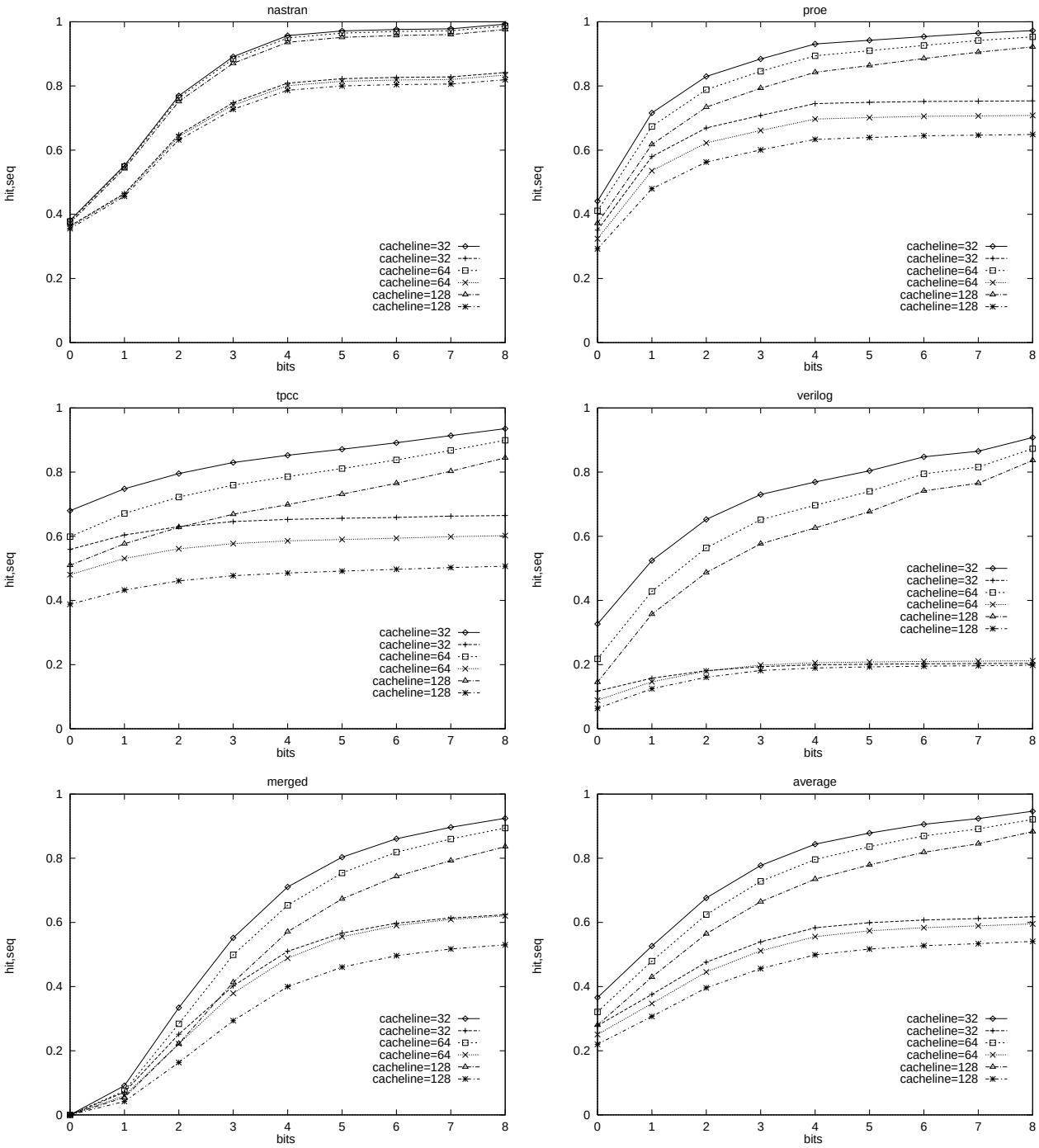

Figure 11: The change in the page mode hit rate and the sequential prefetching hit rate as the cache line size changes. The 'merged' case does not track the average of the four applications because the relative number of transactions per application changes as the cache line size changes.

memory pages open at a time. This both reduces the number of page misses significantly, decreasing the page mode penalty, and reduces the complexity of the memory controller by reducing the amount of state and number of state machines. In fact, only eight (for a uniprocessor) to 32 (for a 8-way multiprocessor) bank controllers are needed even for a 256 bank memory system. Secondly, we implement bank-based prefetching, which further reduces memory latency by speculatively moving the next sequential line into the bus drivers.

The primary results of our investigation are these:

- Using xor bank indexing is slightly better than using the optimal permutation scheme.
- A memory system design should have more than two banks per CPU, and about two to four bank controllers per CPU, in order to make best use of page mode.
- Beyond four bank controllers per CPU, the performance advantage gained is minimal.
- Using a small number of prefetch buffers (two to four per CPU) can significantly improve memory latency further; for our basic workloads, most references were satisfied directly out of the prefetch buffers. Doing this eliminates all DRAM latency from the majority of references.
- Increasing the number of memory banks while keeping the number of memory controllers constant improves performance. This is best done by using an associative arrangement of bank controllers, rather than just conjoining banks in some fixed way.
- Using an associative prefetch cache rather than bank-based prefetching can improve the prefetch rate a bit more.
- Refresh has a negligible impact on the performance of a page mode memory controller.
- These techniques work best for a system with smaller cache line sizes, but provide significant benefit even with large (128 byte) cache line sizes.

13 Acknowledgements

Gheith Abandah and Milon Mackey assisted me greatly by modifying their tools and generating the TPC-C trace for the 'bad workload' case.

14 References

[TR96] Rokicki, Tomas: Indexing Memory Banks to Maximize Page Mode Hit Percentage and Minimize Memory Latency. Hewlett-Packard Laboratories Technical Report HPL-96-95, June 1996.