DRAM Caching

Wayne A. Wong and Jean-Loup Baer
Department of Computer Science and Engineering
University of Washington
Seattle, Wa 98195-2350

e-mail {waynew,baer}@cs.washington.edu

UW-CSE-97-03-04

Abstract

This paper presents methods to reduce memory latency in the main memory subsystem below the board-level cache. We consider conventional page-mode DRAMs and cached DRAMs. Evaluation is performed via trace-driven simulation of a suite of nine benchmarks.

In the case of page-mode DRAMs we show that it can be detrimental to use page-mode naively. We propose two enhancements that reduce overall memory latency in this case: one is the remapping of address bits and the other is selective usage of page-mode under the control of the memory controller.

In the case of cached DRAM we quantify the improvements that can be attained by introducing some SRAM cache on the DRAM chip. We evaluate various design alternatives for the line size and associativity of the SRAM cache.

1 Introduction

As the gap between processor and memory widens, tolerating memory latency has become the biggest challenge to achieving high performance in current microprocessor systems. The great majority of the latency hiding techniques, either based on hardware enhancements or software optimizations, that have been proposed or implemented to date have been directed towards the top of the memory hierarchy, namely, caches at various levels. Generally, main memory has been viewed as having uniform access latency. This is however an over simplification which does not take advantage of developments in memory technology.

Main memory is usually composed of dynamic random-access memory (DRAM) devices [6]. Current DRAMs have an optimized feature called page-mode. In page-mode, the device has an on-chip buffer, that provides access which is a factor of two or more faster than a normal DRAM access[1]. However, as we explain later, page-mode operation lacks flexibility and can result in performance degradation rather than improvement. This is because page-mode is meant to be a high-bandwidth mechanism and its buffer is not designed to be used as a conventional cache.

Among the emerging DRAMs (synchronous DRAMs, EDODRAMs, etc.), two that are particularly attractive for caching are cached DRAMs (e.g., CDRAM and EDRAM [2, 12]). Both devices replace the page-mode buffer with static RAM (SRAM). In addition to providing higher bandwidth, the SRAM can be used as a conventional cache to reduce memory latency [5].

The thrust of this paper is two-fold. First, we investigate methods of reducing memory latency using page-mode DRAMS in a selective manner. Second, we explore the various organization parameters of SRAM cache design for a cached DRAM.

The remainder of the paper is organized as follows. In Section 2, we review the operations of page-mode DRAM (including the reason why following page-mode blindly can be counter-productive) and of cached DRAMs. The methodology that we use to evaluate the effectiveness of the enhancements that we propose is described in Section 3. Section 4 presents the results of two techniques that can be applied to page-mode DRAMs to improve their performance: one is related to the mapping of addresses in the page buffer; the other considers memory controller schemes that select which banks operate in page-mode and which do not. Section 5 investigates the impact of the SRAM cache in DRAM devices and evaluates several design alternatives. Finally, in Section 6 we conclude and place this study in the perspective of integrated processor/memory systems [3, 14].

2 Page-mode DRAM and Cached DRAM Operation

In this section, we review very briefly the operation of page-mode DRAMs and cached DRAMs.

2.1 Page-mode DRAM

An access in DRAM devices usually consists of a row access followed by a column access (see Figure 1). A read request consists first of a row access, reading a row of bits (a DRAM page), containing the desired data bit, from the DRAM array into a page buffer. Second, a column access selects the desired data bit from the page buffer. Assuming that the device has been *precharged*, the latency is the sum of the row access and column access latencies. For typical DRAMs, this can range from

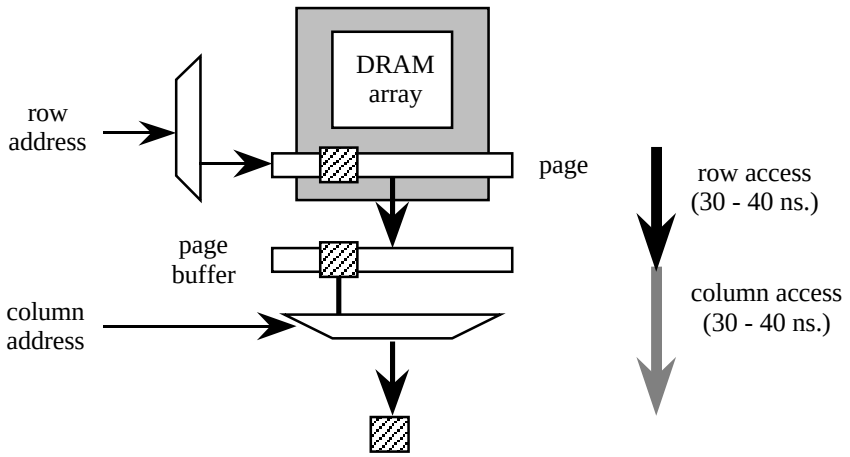

Figure 1: *DRAM operation. This figure shows how a data bit is read from the DRAM device. First, the row address selects a DRAM page that is read into the page buffer. Second, the column address selects the desired data bit from the page.*

30 to 40 nanoseconds for each of the row access and column access latency. Before data can be read from the DRAM array, it has to be precharged. The precharge latency dictates the minimum time between array accesses and can be hidden if there is sufficient time between accesses. Typical DRAM precharge time is 30 to 50 nanoseconds.

In *page-mode* [6], row accesses can be eliminated on successive requests if the desired bits are in the page buffer. The access time is then reduced to the column access time. Thus, the page buffer can be used as a cache with a single long line. However, if the request misses in the page buffer, a whole cycle of precharge, row access, and column access takes place. The precharge is necessary because the DRAM cannot be both precharged and in page mode.

Using page-mode can provide lower latency and higher bandwidth from the memory system. However, as explained in the previous paragraph, using page-mode can also be detrimental to performance. If we assume that precharge, row access, and column access take the same time, say t, then for a series of n accesses, where n is the page buffer size divided by the data size, we have:

- The best case for page-mode operation is $n-1$ hits in the page buffer after the initial miss. The page-mode time would then be: $3 \times t + (n-1) \times t$. The worst case for page-mode operations is a miss on each request. The time for n requests would be $3 \times n \times t$.
- Without using page-mode, the time would be between $2 \times n \times t$ (assuming there is enough time between requests to hide the precharge time) and $3 \times n \times t$ when there is no time between requests.

From the above analysis, it is clear that page-mode operation will be beneficial if the hit rate of the page buffer is above 50%. Of course, the 50% threshold is only an approximation that depends on the relative timings for precharge, row access, and column access as well as on the demands, reads or writes from the processor. In particular, if the write latency can be hidden, e.g., with the help of

benchmark	description
cholesky	The Cholesky factorization component of the Nasa7 floating point benchmark.
compress	Compress a 1 MB file using adaptive Lempel-Ziv coding.
fft	The FFT component of the Nasa7 floating point benchmark.
gcc	GNU C compiler which produces Sun3 assembly code
geometry	Sets up arrays for a vortex method solution and performs Gaussian elimination on the resulting arrays.
hydro2d	Solves hydrodynamic Navier Stokes equations
su2cor	Computes masses of elementary particles
tomcatv	Vectorized mesh generation program.
vpenta	Inverts 3 matrix pentadiagonals.

Table 1: Benchmark descriptions

write buffers and a write-back policy in the processor cache, then the read requests to the DRAM must have a page hit rate above 50% for page-mode to be effective. What this simplified analysis shows though is that page-mode operation will be effective for access to large data sets, overflowing or bypassing caches, with low stride between consecutive accesses to individual DRAM banks. This addressing pattern is typical of vector applications [11, 7]. However, page-mode operation can be detrimental for applications accessing memory "randomly". We will return to this problem and present some solutions in Section 4.

2.2 Cached DRAMs

In newer DRAM technologies such as CDRAM and EDRAM [2, 12, 5], the page buffer is replaced with a small SRAM cache. The design of these DRAMs encourage on-chip DRAM caching and eliminate the drawbacks of page-mode DRAMs. The first advantage of these devices over page-mode DRAMs is that the use of the SRAM enables the simultaneous precharging of the DRAM array and access of the SRAM cache. Thus, according to our model of the previous section, access to the cache will take time t, while a cache miss will take between $2 \times t$ and $3 \times t$. Second, instead of having a single long cache line, the cache can be organized with parameters more like a traditional cache. For example, the cache in the CDRAM has 256 entries [13]. Moreover, with separate address lines (for the DRAM array and the SRAM cache) and external tag logic, the CDRAM's cache can be made set-associative. In Section 5, we explore the design space of the SRAM cache.

3 Methodology

To evaluate the effectiveness of DRAM on-chip caching, we use trace-driven simulation. The traces were generated from an Alpha 21064 using the ATOM [16] software instrumentation tool. ATOM inserts instrumentation code in the object code. The generated references are those of an uninstrumented binary. In our study we concentrate only on data references. For our benchmarks, the instruction reference working set fits well in a small instruction cache and would generate few memory references beyond the compulsory misses [10].

Figure 2: *Workload misses per instruction. This figure shows the memory system load presented by the benchmarks. For each benchmark, the number of misses per 100 instructions are shown for two direct-mapped cache (8 KB and 256 KB). The height of the bar is the MPI on an 8 KB cache. The lower component of each bar is the MPI on a 256 KB cache.*

Our workload consists of benchmarks from SPEC92 [4] that exhibited significant data reference misses per instruction (MPI). The benchmarks are described in Table 1. The MPI for the benchmarks are shown in Figure 2 for two caches of respective sizes 8KB and 256 KB, both direct-mapped and with a line size of 32 bytes.

3.1 Simulated system

Our interest in this paper is on the main memory subsystem. Therefore, we model the cache hierarchy as a single-level cache[1]. This cache uses a write-back policy which minimizes memory traffic [6] and which is the one most often used in caches closest to main memory. The processor is modelled with an ideal pipeline, i.e., each instruction executes in a single cycle, except when there is a cache miss. In this case, the memory latencies depend on the memory subsystem being modelled as explained below.

We have performed our experiments with two "extreme" cache sizes: 8 KB and 256 KB. The small 8KB capacity corresponds to low-end machines, e.g. the MicroSparc as indicated in [14]. The small capacity cache can also be seen as a way to model the behavior of systems with larger caches running applications whose working set sizes are larger than those we are using. The larger capacity 256 KB cache corresponds to higher end systems. Although larger caches, in the megabyte range, are available, we limited ourselves to 256 KB in order to have meaningful data for the applications we simulated. Both caches are direct-mapped with a 32 byte line size.

The memory subsystem, shown in Figure 3, has four banks each with an 8 KB page buffer(or cache) for an aggregate of 32 KB of page buffer (or cache). The amount of page buffer is derived from a

[1]We will call this cache the *board-level* cache when there is a risk of confusion between it and the DRAM cache.

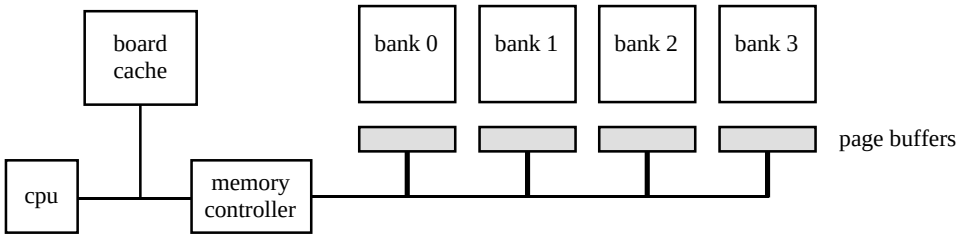

Figure 3: *Simulated memory system. This figure shows the experimental memory system. The memory subsystem has four banks. There is an aggregate of 32 KB page buffer (8 KB per bank). The memory controller determines the mapping and the operating mode of each bank.*

32 MB system composed from 64 4×1 Mb DRAMs. We assume that a board-level cache miss can be satisfied with one memory access. Banks are interleaved on a modulo(8 KB) basis.

As explained in Section 2, there are three possible access latencies for a main memory access, namely: page-mode, precharged, and random. When the DRAM is in page-mode, *page* is the latency of a page-buffer hit. We chose a simulated processor clock speed of 166 MHz (6 ns. cycle). For a 30 ns. page-buffer hit, this is 5 cycles. *Random* is the latency of a miss where the full penalty of precharge, in addition to the row and column access latencies, is seen and is 15 cycles. *Precharged* is the latency of a non-page-mode DRAM access where some of the precharge latency is hidden. It includes the row and column access latencies and is 10 to 15 cycles, depending upon how much of the precharge time (5 cycles) is hidden.

3.2 Memory performance metrics

We use two metrics to evaluate the effectiveness of the DRAM caching schemes. The first one is the page-buffer miss rate. The miss rate indicates the effectiveness of using the relatively small amount of DRAM cache. For the cached DRAMs, an increase in hit rate will be directly correlated to a reduction in memory latency. For page-mode DRAM, however, a decrease in page misses is only one of the factors that lead to a reduction in memory latency. The metric of interest is the number of cycles that the processor waits for a memory request to complete. Since our goal is to evaluate the effectiveness of the memory subsystem below the board-level cache, we will account only for the cycles needed to transfer data between the board-level cache and the DRAM. We therefore use as the metric, memory cycles per instruction in the memory subsystem ($MCPI_s$), defined as:

$$MCPI_s = \frac{Cycles\ waiting\ for\ memory\ requests}{total\ number\ of\ instructions}$$

4 Enhancing the Performance of Page-mode DRAMs

As explained in Section 2, page-mode DRAMs have the ability to cache data. However, the cache is severely constrained. It consists of a single long line (one per memory bank) and the data stored in the line corresponds to consecutive byte addresses. If accesses to the DRAM memory exhibit

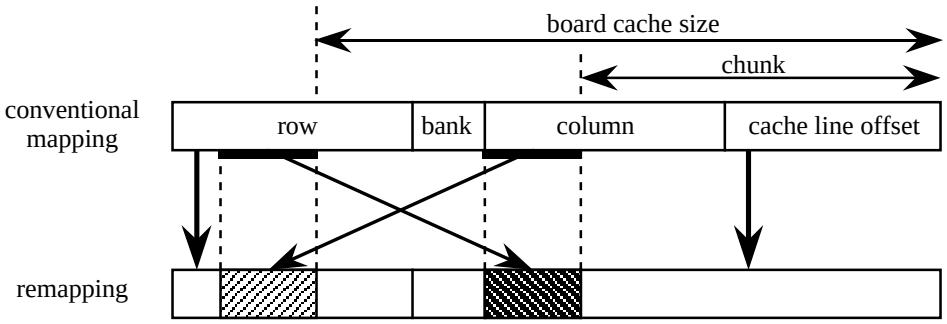

Figure 4: *Memory mapping. This figure shows how the memory request address is remapped.*

good spatial locality, e.g., accessing all the elements of a vector sequentially, then the hit rate in the DRAM cache will be high and operating in page-mode will be beneficial. When the hit rate is low, a common occurrence because of the filtering effect of the board-level cache, then not using page-mode and attempting to hide the precharge time is preferred.

In this section, we investigate two ways to enhance the performance of page-mode DRAMs: first, we examine the impact of address mapping so that the contents of the page buffer can be divided in several subsets, and, second, we look at schemes to selectively control the page-mode operation on a per bank basis.

4.1 Memory mapping

As shown in Figure 1, the conventional page-mode DRAM has a set of row address lines and column address lines. An address generated by the CPU and resulting in a cache miss will be seen by the DRAM as having 4 components:

- The (board-level) cache block offset. For example, in our modeled system with 32 byte cache lines, this would be the low 5 bits, bits 4-0.
- The remaining bits are split in three fields: bank selection, row address, and column address. In a conventional DRAM (see Figure 4), the low order bits correspond to the column address, the next bits are used for bank selection, and the remaining upper bits constitute the row address.

In the conventional mapping of addresses, the choice of row and column addresses can be far from optimal for a memory system with a board-level cache. The column address bits are bits that correspond to low order bits of the index field for the board-level cache. Consequently, any conflict miss in the board-level cache, which is most always larger than the DRAM cache, will certainly result in a page miss in the DRAM page buffer. One way to circumvent this problem is to use other bits of the address for the DRAM row and column addresses. In the Alphastation 600 [17], the low order column bits are exchanged with the row bits and the high-order bits are used for bank selection. Now both the (board-level) cache line and the one it has just replaced can be cached

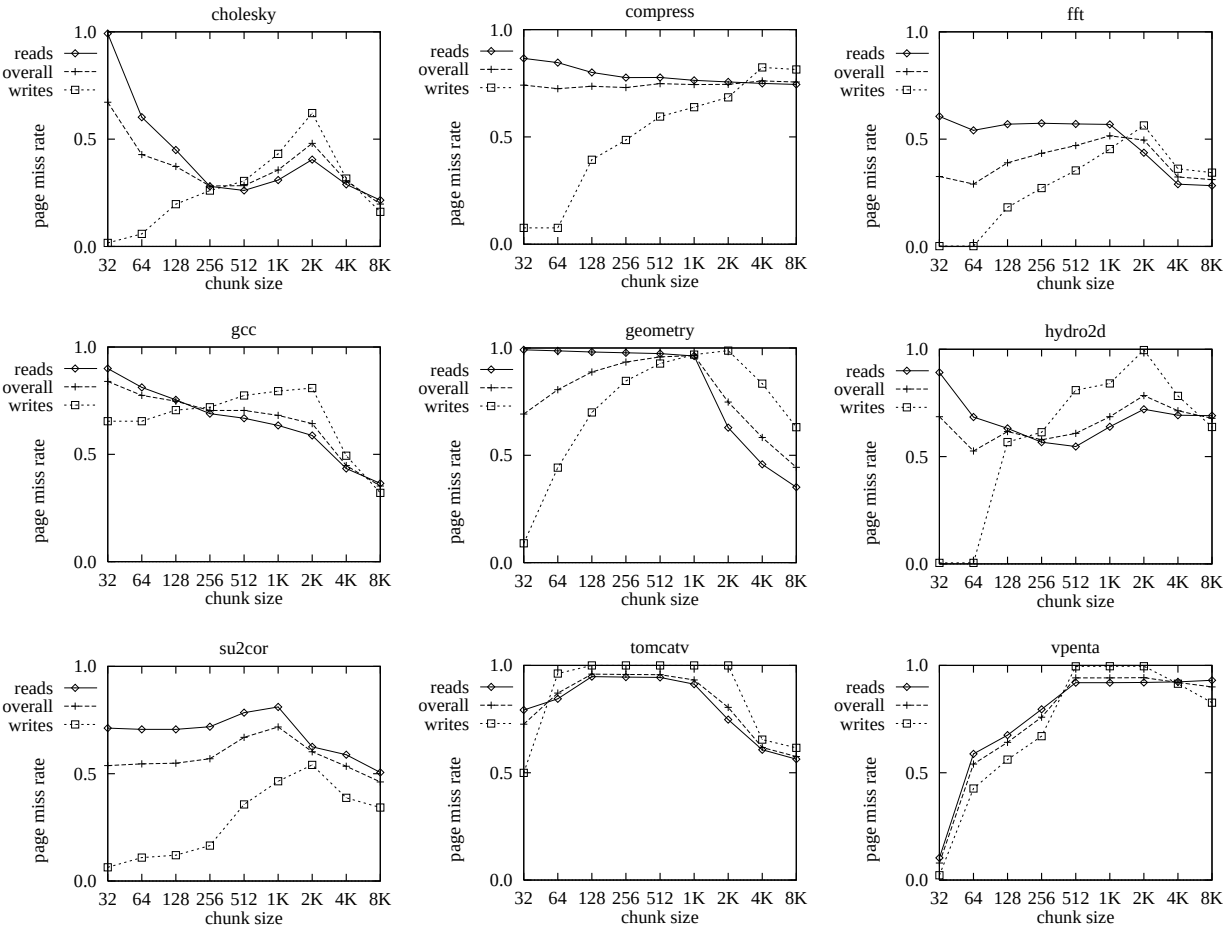

Figure 5: *Page buffer interleaving with an 8 KB board-level cache. These graphs show the page buffer miss rate for various page buffer interleavings with an 8 KB board-level cache. The overall miss ratio is the number of misses to the number of references. The read miss ratio is the number of read misses to the number reads. The write miss ratio is similar to the read miss ratio.*

by the same DRAM row. However, this mapping destroys the spatial locality that existed in the DRAM buffer and will be detrimental to page-mode operation for sequential accesses. For the Alphastation 600, the remapping begins at bit 8, mapping only 256 bytes contiguously.

We use a similar method but first we want to determine which bits should be swapped in order to find a good compromise between eliminating a majority of page buffer misses due to conflict misses in the board-level cache while preserving some spatial locality. To that effect, we varied the number of bytes mapped contiguously, that we call *chunks*, from the minimum of a board-level cache line (32 bytes) to a complete DRAM page (8 Kbytes). Addresses of consecutive chunks in the DRAM cache were taken modulo the board-level cache size (cf. Figure 4).

The ensuing DRAM page buffer miss rates are plotted in Figures 5 and 6 for the various applications[2] and for the two different board-level caches. For each application, we plot miss rates vs. the chunk size. The three miss rates of interest are:

[2]We removed the plot for *gcc* in the 256 KB case, since the board-level misses were insignificant.

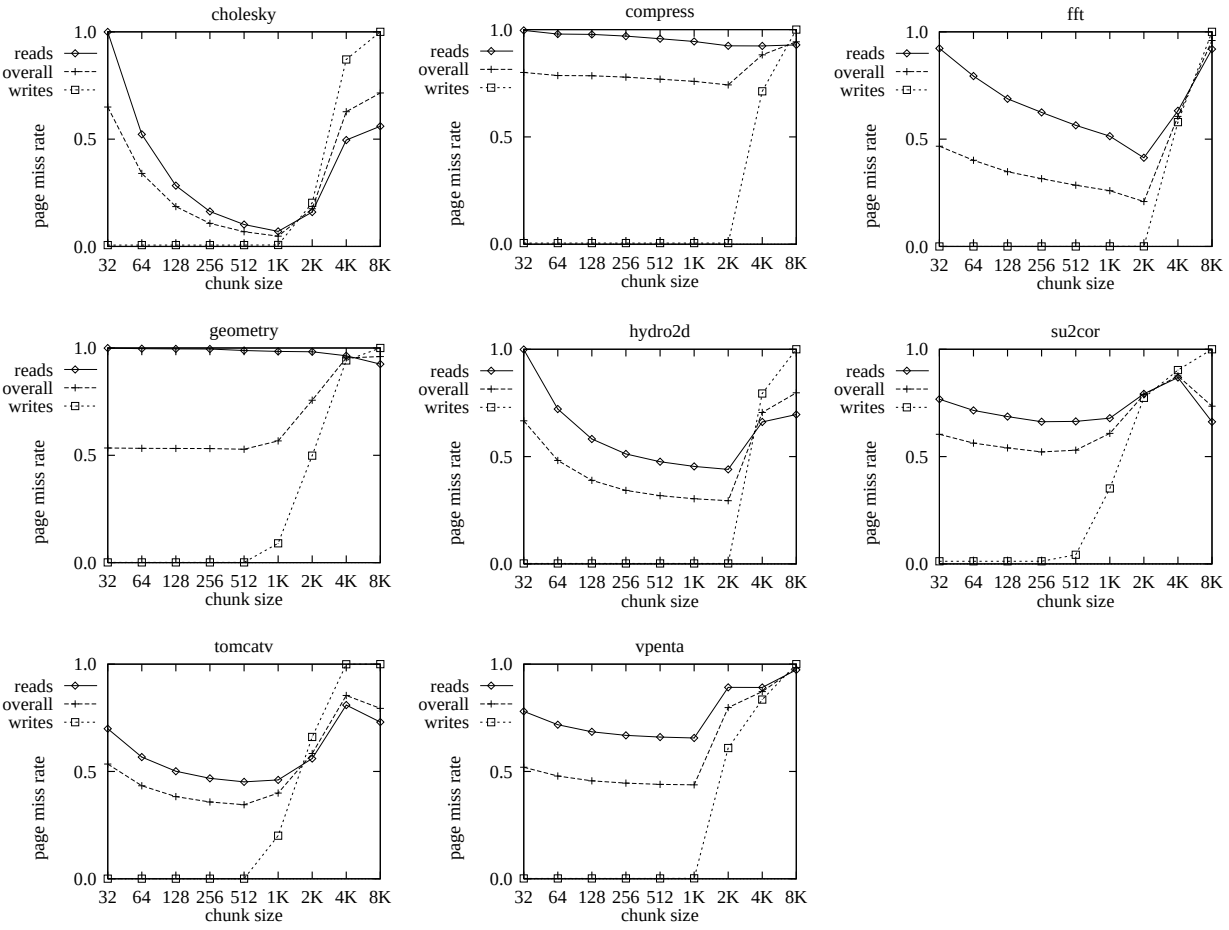

Figure 6: *Page buffer interleaving with a 256 KB board-level cache. These graphs show the page buffer miss rate for various page buffer interleavings with a 256 KB board-level cache. The overall miss ratio is the number of misses to the number of references. The read miss ratio is the number of read misses to the number reads. The write miss ratio is similar to the read miss ratio.*

- The overall page miss rate, i.e., $\frac{number\ of\ page\ misses}{number\ of\ DRAM\ accesses}$
- The read miss rate, i.e., $\frac{number\ of\ read\ page\ misses}{number\ of\ DRAM\ read\ accesses}$
- The write miss rate, i.e., $\frac{number\ of\ write\ page\ misses}{number\ of\ DRAM\ write\ accesses}$

In the case of the small board-level cache, the amount of page buffer (32 KB total) is larger than the cache's capacity and acts as a second-level cache. Having long buffer lines will help in the case of capacity misses while having short lines will be more efficient to reduce the latency due to conflict misses. As can be seen from Figure 5, overall miss rates and read miss rates are generally lowest for chunks of size 8 KB, the maximum chunk size. (*vpenta* is a glaring exception.) Write miss rates are less important for overall performance for two reasons: (i) writes are due to replacement of dirty lines and are less frequent than reads, and (ii) write-backs are in the background and do not stall the processor if subsequent read requests are sufficiently far away. Looking at Figure 5 and choosing 8 KB chunks, we would expect *cholesky* and *fft* to benefit handsomely from page-mode operation

while the performance degradation in *vpenta*, *compress* should be severe and the degradation of *hydro2* could be moticeable.

In the case of the 256 KB cache, the choice for the best chunk size is significantly different. Not doing any remapping, i.e., leaving an 8 KB chunk, could lead to severe losses in page-mode operation since for that chunk size the miss rates are often close to 100% and always larger than 50%. We would not expect any application to benefit from page-mode without remapping. Looking at Figure 6, the minimum overall and read miss rates are obtained for a chunk size between 512 bytes and 2 KB for all applications. These sizes are a compromise between removing more conflict misses (small size) and keeping spatial locality. Also, we note that the write miss rates drop to almost 0 for all applications except *su2cor* and *tomcatv* when the chunk size is less than 2 KB. This is because some of the victim and requested lines map to the same DRAM row. By choosing a chunk size of 1 KB, the latencies of board-level cache misses in *cholesky* and *fft* should be greatly reduced in page-mode operation. On the other hand, those from *compress* should still be worse. For the other applications, the end result might depend on whether the write miss latencies can be completely hidden or not.

In the remainder of the paper, we choose a chunk size of 1 KB for remapping with the 256 KB cache and not to remap with the 8 KB cache.

4.2 Evaluation of remapping

Figure 7 shows the relative performance of page-mode operation versus precharge for the three cases of: 8 KB board-level cache and 8 KB chunk, 256 KB board-level cache and 8 KB chunk, and 256 KB board-level cache and 1 KB chunk. For the time being, we only look at the leftmost bar (for each application) that gives the ratio:

$$\frac{MCPI_s \ for \ page \ mode \ operation}{MCPI_s \ for \ precharge \ operation}$$

In *page-mode* operation, a DRAM cache hit returns data in *page-mode*, i.e., 5 cycles. A miss takes *random*, i.e. 15 cycles. In the *precharge* scheme the DRAM is always precharged after the access. All accesses take *precharged*, i.e., between 10 and 15 cycles. The *precharge* scheme does not use page-mode and is our baseline. In contrast, the *page-mode* scheme always leaves the DRAM in page-mode after each request, expecting that the next request will be in the page buffer.

In general, the results correlate well with our predictions from the last section. In the case of the 8 KB board-level cache, *cholesky* and *fft* benefit significantly from page-mode operation while *compress, hydro2d* and *vpenta* suffer from it. *gcc, geometry* and *su2cor*, with overall hit ratios close to 50%, perform slightly better in page-mode and there is no difference for *tomcatv* with a miss ratio close to 60%.

In the case of the 256 KB board-level cache and no remapping, page-mode operation is always worse, to a varying degree, than precharge. This is consistent with the data of Figure 6.

Finally, in the remapping case, 256 KB board-level cache with 1 KB chunk, page-mode operation works very well for *cholesky* and *fft* and badly for *compress*, as expected. For the other applications, page-mode operation is slightly more beneficial than the crude 50% threshold would have led us to believe.

Figure 7: *Page-mode MCPIs. This figure shows the normalized performance of the various page-mode schemes and the effect of remapping. The MCPIs are normalized to the performance of the precharge page-mode scheme.*

4.3 Selectively using page-mode

While remapping clearly leads to better page-mode operation, there are applications that would benefit if page-mode operation could be turned on or off depending on the memory reference patterns. We therefore investigate memory controller schemes that use page-mode selectively. Whether to operate in page-mode or not is based on the history of past accesses to the DRAM. The schemes are summarized in Table 2.

scheme	description	hit	miss
precharge	The DRAMs are always precharged after satisfying the current request	*precharged*	*precharged*
page − mode	Page-mode is always used.	*page*	*random*
mru	The two most recently used banks are kept in page-mode. The other banks are precharged.	*page* or *precharged*	*random* or *precharged*
*affinity*1	The bank is put in page mode after two consecutive read accesses to the same page.	*page* or *precharged*	*random* or *precharged*
*affinity*2	Strong two-bit counter scheme based on read accesses	*page* or *precharged*	*random* or *precharged*

Table 2: *Page-mode control schemes.* This table describes each of the page-mode schemes. Hit is the latency of a request that would hit in the page buffer. Miss is the latency otherwise.

We have already described the operations of the *precharge* and *page-mode* schemes. Similar to *page-mode* is *mru*. However, *mru* leaves only the two most recently used memory banks in page-mode. If there is a correlation between inactive banks and page buffer misses in inactive banks, then *mru* will do better than *page-mode* by precharging inactive banks. The two other schemes, *affinity1* and *affinity2*, attempt to predict whether to leave the DRAM bank in page-mode or to precharge. Similar to simple branch prediction logic [9] *affinity1* and *affinity2* use a 1-bit and 2-bit, respectively, prediction scheme as to whether to use page-mode or precharge. Incrementing or decrementing the prediction counter is done on read accesses only.

Figure 7 shows the performance of the above page-mode schemes in the same 3 configurations as in the remapping evaluation. For each application, we have shown the performance of the selective schemes relative to the *precharge* scheme. As in the page-mode vs. precharge comparison, if the relative $MCPI_s$ is above 1, then the application would be better off simply precharging after each request.

The *mru* (second bar from the left) scheme presents no advantage over *page-mode* for the 8 KB case. Sometimes, the performance is better (*compress, geometry*), sometimes it is markedly worse (*cholesky, su2cor*). For the larger cache, the *mru* scheme corrects some errors of the *page-mode* scheme, i.e., some banks are precharged rather than exhibiting page buffer misses. *mru* performs better than *page-mode* whether there is remapping or not.

Of the two *affinity* schemes (the two bars on the right), *affinity2* performs best. In the 8 KB case, it is the scheme of choice since all applications with this scheme behave as well or better than *precharge*. Under this scheme, when there is no locality, the banks will remain in the *precharged* mode. When there is sufficient locality, they will be in *page-mode*. The only time where there might be a slight loss, with respect to *mru* or *page-mode*, is for the "learning curve" of the 2-bit counter that keeps a bank in precharge mode for 1 or 2 references too many. In the 256 KB case

with no remapping, *affinity2* yields the best results, always performing better than both *precharge* and *page-mode*. When remapping is implemented, *affintiy2* will always be better than *precharge*. In two applications, *geometry* and *vpenta*, operating in *page-mode* would be better. Although, it is not difficult to find pathological cases where this behavior can be exhibited, further analysis of the reference patterns is needed before we can explain this "anomalous" behavior. This might be due to short bursts of "read,write" sequences to the same page picked up by *page-mode* but initially discarded by *affinity2*.

4.4 Summary

In this section we have shown that using page-mode operation in DRAMs in a straightforward fashion could degrade performance over simply precharging banks after each access. This degradation would get relatively worse as the board-level cache gets bigger. Two factors that can enhance the operation in page-mode are: (i) the mapping of addresses to row and address lines in the DRAM so that the hit rates in the page buffer are higher, and (ii) selective use, monitored by the memory controller, of the page-mode operation itself. With these enhancements, page-mode operation will always be benificial with up to 100% improvement.

5 Cached DRAMs

In the previous section, we evaluated the efficacy of the caching effects that are possible with page-mode DRAMs. In this section, we investigate the effectiveness of an SRAM cache on the DRAM chip. Recall that in these cached DRAMs, the page buffers of page-mode DRAMs are replaced with SRAM and that, consequently, one definite advantage of the cached DRAM is the possibility of simultaneous precharging of the DRAM array and access to the SRAM cache. We first quantify this advantage by comparing cached DRAM latencies with those of page-mode DRAM from Section 4. We then investigate various design organizations for the SRAM cache.

5.1 SRAM advantage

One of the disadvantages for caching with page-mode DRAMs is the guessing involved in deciding whether to precharge the DRAM or leave it in page-mode. While the schemes from Section 4.2 can be used to improve the decision heuristic, they do not work well in all cases and they add to the complexity of the memory controller. The replacement of the page buffer with SRAM eliminates this guessing: accessing the SRAM cache does not preclude the simultaneous precharging of the DRAM array.

The advantage of the SRAM cache over page-mode DRAM is shown in Figure 8. We compare the $MCPI_s$ of the cached DRAM against two page-mode DRAM schemes from section 4.2, *page-mode* and *affinity2*. For the cached DRAM, we use the same latencies as the page-mode DRAM, i.e., 5 cycles for a hit and 10-15 cycles for a miss depending on whether precharging was finished or in progress. For the 256 KB cache, we use remapping with chunk 1 KB. The $MCPI_s$ are normalized to the $MCPI_s$ of the *precharge* DRAM scheme as in Figure 7.

From the graphs, we directly see the benefit of being able to simultaneous precharge and access

Figure 8: *Comparison of cached and page-mode DRAM. For each benchmark, the MCPIs (normalized to the precharge scheme with page-mode DRAM) are shown for cached DRAM against two schemes with page-mode DRAM.*

the DRAM cache. The cached DRAM (the rightmost bar) has lower latencies than all the other schemes with page-mode DRAMs. When page-mode was efficient, e.g,. with *cholesky* and *fft*, the gains are minimal. When *page-mode* and/or *affinity2* worked badly, the relative gains are more impressive (*geometry, su2cor*).

5.2 Line size and associativity

We now investigate the design parameters for the SRAM cache with the global constraint of fixed capacity, i.e., 8 KB per bank. Many studies have been performed to characterize the best choice of line size and associativity for a given cache capacity (see, e.g., [15]). The parameters for the design of the SRAM cache on the DRAM chip could be completely different since this cache does not receive requests comparable to those of a (processor) cache.

The SRAM cache on the DRAM chip will receive 3 types of requests.

- Capacity misses if the board-level cache is small. The SRAM cache will play the role of a (small) second-level cache.

- Conflict misses in the board-level cache. The SRAM cache could play the role of a victim cache [8].
- Sequential accesses to large data structures. The SRAM cache could play the role of a stream buffer [8].

In our simulations, we varied the line size from 8 KB (the original one) to 128 bytes and we varied the associativity from direct-mapped (the original one) to 4-way (of course, only when the capacity allowed us to do it). Figure 9 shows the results for the 8 KB cache and Figure 10 for the remapped 256 KB cache. In both figures the $MCPI_s$ are normalized to the performance with a single line of 8 KB (direct-mapped).

For the 8 KB cache, we did not know what to expect since the three types of requests above do exist. The first observation is that for all applications, increasing the associativity for a given line size will result in better performance. For example, when we fix the line size at 1 KB, we see improvements of about 30% in *hydro2, su2cor, tomcatv* and *vpenta* when mvoing from direct-mapped to 4-way set associativity. While this line size is not the best choice for *fft* the end result is still a performance improvement of 15% over the best choice of line size and a direct-mapped SRAM cache. The only exception is for *geometry* where sequential accesses dominate: the longer the line the better the performance.

For the 256 KB cache, there should be almost no capacity misses, except for *compress*, and the remapping already provides a limited form of associativity. Thus, the advantages of more explicit associativity should not be as important. When we have a 1 KB line and a 4-way set-associative SRAM cache, three applications (*hydro2d, su2cor* and *tomcatv*) benefit from associativity but less than in the 8 KB case and other applications such as *fft* and, to a lesser extent, *vpenta* perform worse.

Although there is no firm conclusion of best associativity or best line size, the common trend is that a limited number of large lines will perform well. This is an encouraging result since it implies that the number of tags to be stored will be small and therefore the SRAM overhead will be limited.

6 Conclusions

Memory latency has become the major bottleneck in the performance of high-end systems. In order to reduce or tolerate this latency, a number of techniques must be used at various points in the memory hierarchy. In this paper, we have focused at the DRAM level. We have presented methods to enhance page-mode DRAMs and explored the design space of cached DRAMs. The schemes that we have proposed have been evaluated on a set of nine benchmarks.

Our first effort has been to provide two methods to enhance the performance of page-mode DRAMs. The first one is to remap the addresses of DRAM requests so that conflict misses in the board-level cache do not result in conflict misses in the page buffer. In that case, the page buffer acts partially as a victim cache. The second is to use adaptive methods to control when banks should be precharged and when they should be left in page-mode. A simple 2-bit predictive method yields definite improvements.

The combination of these two methods always insure that the page-mode DRAM will be more efficient than not using page-mode in standard DRAMs. Improvements vary from insignificance,

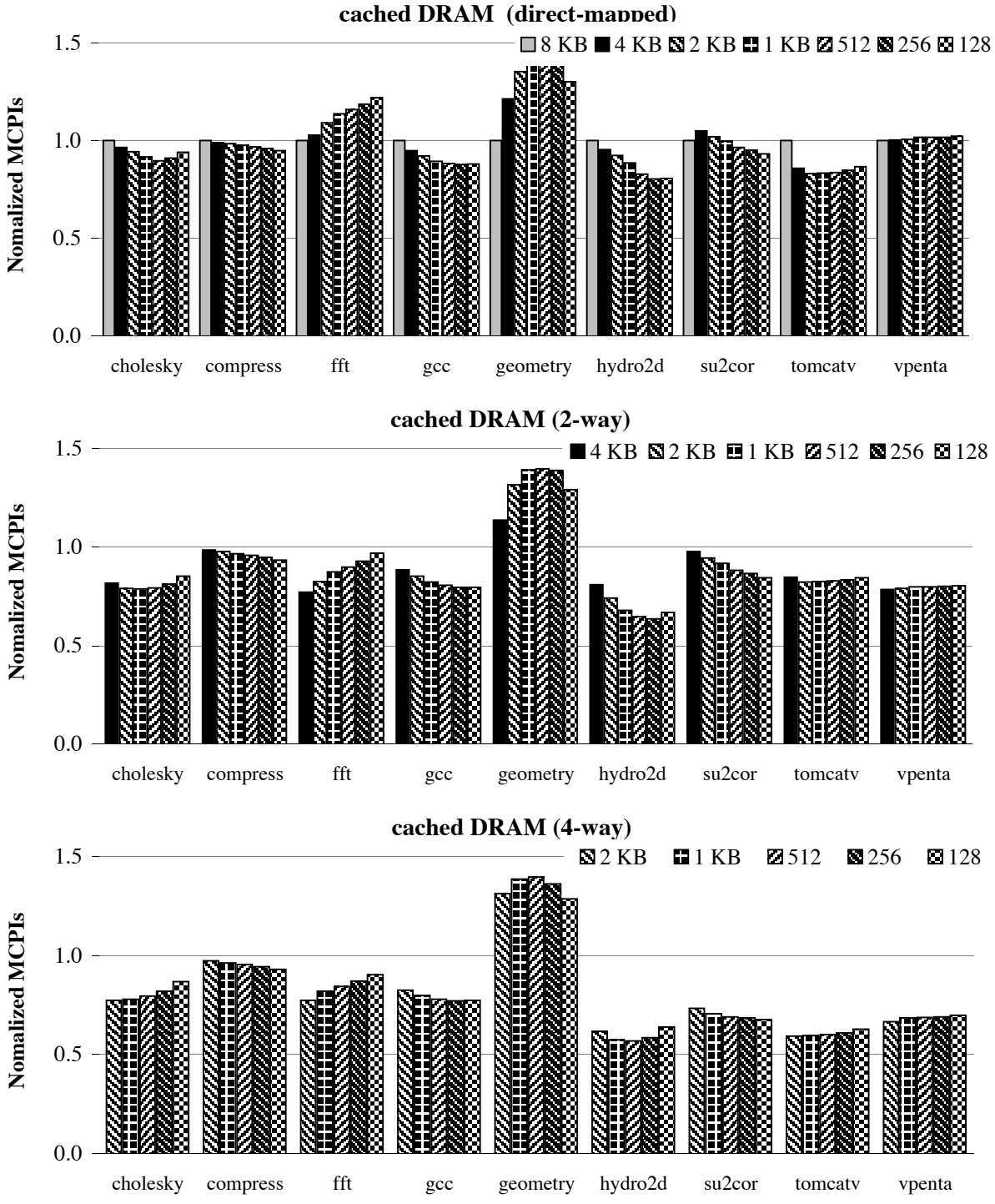

Figure 9: *Cached DRAM organizations with an 8 KB board cache. This figure shows the normalized MCPIs for various DRAM cache line sizes and associativity. The MCPI is normalized to the direct-mapped cached DRAM with 8 KB lines. The board cache is 8 KB and the per bank DRAM cache is 8 KB.*

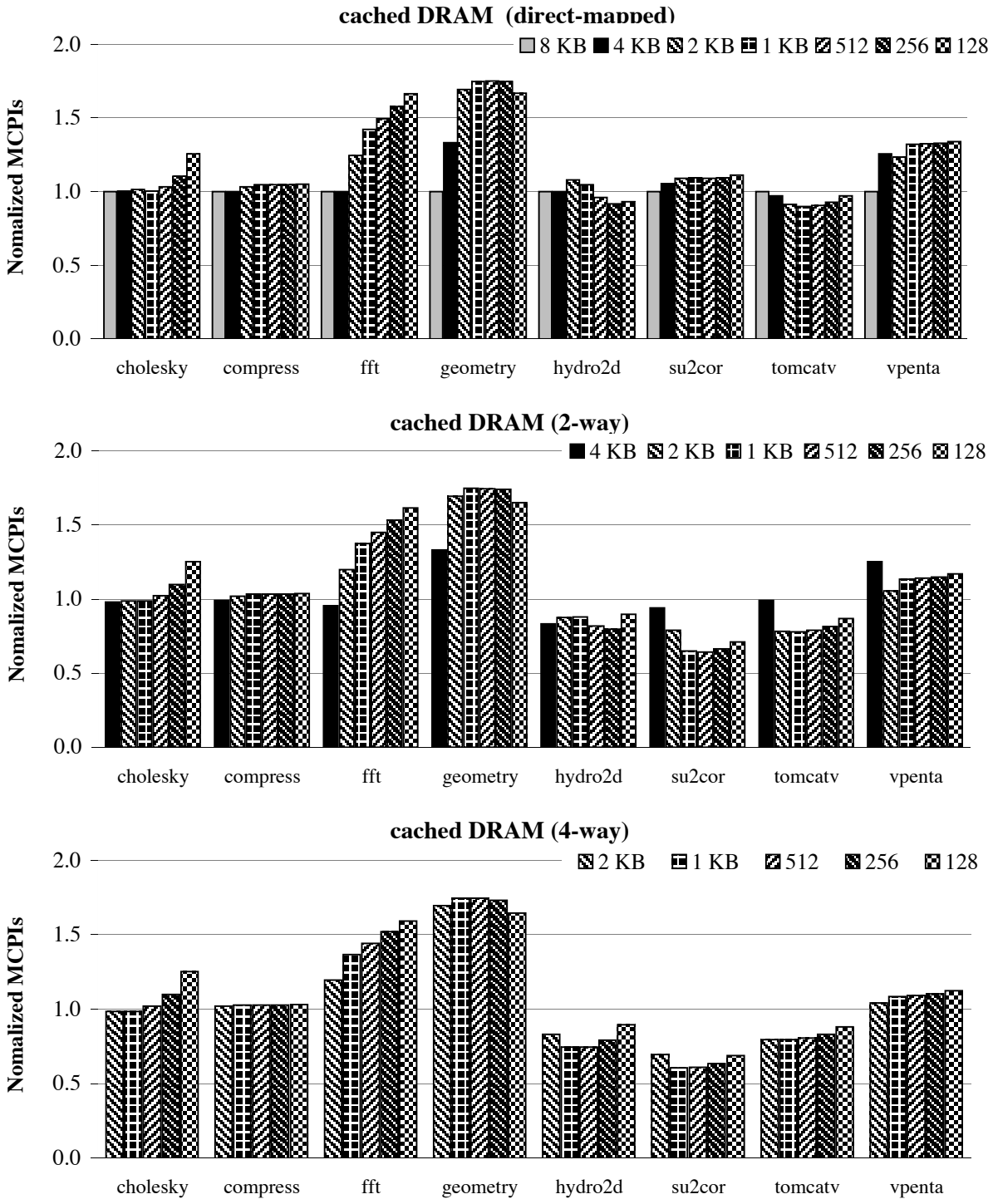

Figure 10: *Cached DRAM organizations with a 256 KB board cache. This figure shows the normalized MCPIs for various DRAM cache line sizes and associativity. The MCPI is normalized to the direct-mapped cached DRAM with 8 KB lines. The board cache is 256 KB and the per bank DRAM cache is 8 KB.*

when the requests are almost all due to lack of capacity in the board-level cache, to a reduction of about half of the memory latency. This is in contrast with a naive use of page mode operation, without remapping and without adaptive control, that results in a degradation of performance for half of the applications when the board-level cache is small and for all applications when the board-level cache is large.

When the page-mode DRAM is replaced by a cached DRAM, the latencies are necessarily always reduced. The reduction in latency is improved by making the SRAM cache more associative either explicitly or implicitly via remapping. However, the prefetching effect of long lines in the SRAM is always important and an implementation with smaller lines can be detrimental. This last result is notable since longer lines reduce the hardware overhead brought upon by the tags.

We can place this study in two contexts. The first one, corresponding to the large board-level cache, indicates that the correct application of page-mode DRAMs and a rather straightforward implementation of a cached DRAM can provide important benefits to all applications. The second relates to the recent proposals on integrating processor and memory on the same chip [3, 14]. If indeed this integration becomes technologically feasible and cost-effective, then the processor, small board-level cache, and the SRAM cache and the DRAMs can all be integrated. What our study shows is that the SRAM can be effective even with a very simple organization.

References

[1] *Texas Instruments MOS Memory Data Book*. 1996.

[2] Dave Bursky. Fast DRAMs can be swapped for SRAM caches. *Electronic Design*, pages 55–56,60–67, July 1993.

[3] Michael Deering, Stephen Schlapp, and Michael Lavelle. FBRAM: A new form of memory optimized for 3d graphics. In *Proc. SIGGRAPH Conference*, pages 167–174, 1994.

[4] Jeffrey Gee, Dionisios Pnevmatikatos, Mark Hill, and Alan Smith. Cache performance of the SPEC benchmark suite. *IEEE Micro*, pages 17–28, August 1993.

[5] Charles Hart. CDRAM in a unified memory architecture. In *Proc. Spring CompCon*, pages 261–266, 1994.

[6] John Hennessy and David Patterson. *Computer architecture: A quantitative approach*. Morgan Kaufmann, Palo Alto, CA, 1990.

[7] W.-C Hsu and James Smith. Performance of cached DRAM organizations in vector supercomputers. In *Proc. 20th Int. Symp. on Computer Architecture*, pages 327–336, 1993.

[8] Norm Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. of 17th Int. Symp. on Computer Architecture*, pages 364–373, 1990.

[9] Johny Lee and Alan Smith. Branch prediction strategies and branch target buffer design. *Computer*, 17(1):6–22, 1984.

[10] Ann Marie Maynard, Colette Donnelly, and Bret Olszewski. Contrasting characteristics and cache performance of technical and multi-user commercial workloads. In *Proc. 8th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 145–156, 1994.

[11] Sally McKee and William Wulf. Access ordering and memory-conscious cache utilization. In *Proc. of 1st Int. Symp. on High-Performance Computer Architec ture*, 1995.

[12] Ray Ng. Fast computer memories. *IEEE Spectrum*, pages 36–39, October 1992.

[13] Steven Przybylski. New DRAMs improve bandwidth (part 1). *Microprocessor Report*, pages 18–21, February 1993.

[14] Ashley Saulsbury, Fong Pong, and Andreas Nowatzy. Missing the memory wall: The case for processor/memory integration. In *Proc. of 23rd Int. Symp. on Computer Architecture*, pages 90–101, 1996.

[15] Alan Smith. Line (block) size choice for CPU cache memories. *IEEE Transaction on Computers*, C-36(9):1063–1075, September 1987.

[16] Amitabh Srivastava and Alan Eustace. ATOM - a system for building customized program analysis tools. In *Proc. of the SIGPLAN Conf. on Programming Language Design and Implementation*, pages 196–205, 1994.

[17] John Zurawski, John Murray, and Paul Lemmon. The design and verification of the AlphaStation 600 5-series workstation. *Digital Technical Journal*, 7(1):89–99, 1995.