A Permutation-based Page Interleaving Scheme to Reduce Row-buffer Conflicts and Exploit Data Locality

Zhao Zhang Zhichun Zhu Xiaodong Zhang
Department of Computer Science
College of William and Mary
Williamsburg, VA 23187
{zzhang, zzhu, zhang}@cs.wm.edu

Abstract

DRAM row-buffer conflicts occur when a sequence of requests on different rows goes to the same memory bank, causing much higher memory access latency than requests to the same row or to different banks. In this paper, we analyze the sources of row-buffer conflicts in the context of superscalar processors, and propose a *permutation-based page interleaving scheme* to reduce row-buffer conflicts and to exploit data access locality in the row-buffer. Compared with several existing schemes, we show that the permutation-based scheme dramatically increases the hit rates on DRAM row-buffers and reduces memory stall time of the SPEC95 and TPC-C workloads. The memory stall times of the workloads are reduced up to 68% and 50%, compared with the conventional cache line and page interleaving schemes, respectively.

1 Introduction

Concurrent accesses to multiple interleaved memory banks are supported in modern computer systems, where each bank has a row-buffer holding a page of data.[1] With the significant improvement in memory bandwidth, the DRAM access speed is becoming more crucial to determine the memory stall time of a program execution [6]. One effective solution to address this issue is to utilize the available concurrency among multiple DRAM banks, and to exploit data locality available in the row-buffer of each DRAM bank. However, conflicting performance benefits exist between exploiting access concurrency and data locality in the row-buffer. Memory interleaving scheme designs directly determine the effectiveness of the solution. A conventional memory interleaving scheme allocates consecutively addressed data blocks to consecutive memory banks using a modular mapping function. The size of the interleaved data block can be a word, a cache line, multiple cache lines, a page, or multiple pages. In general, using larger interleaved data blocks leads to more data locality in each DRAM row-buffer but lower concurrency among the multiple banks.

Regarding the efforts of exploiting locality, people have proposed techniques to take advantage of the row-buffer, which serves as a natural "cache" with a large block size. Some DRAM manufacturers even add SRAM caches into the DRAM chips. With the improvement of DRAM row-buffers in the accumulative size, exploiting row-buffer locality is becoming more and more effective for memory system performance improvement. One major bottleneck limiting this effort comes from DRAM row-buffer conflicts which occur when a sequence of requests on different pages goes to the same bank, causing conflict misses in the row-buffer. Frequent row-buffer misses can significantly increase access latency and degrade overall performance. Compared with a row-buffer hit, a row-buffer miss may cause additional DRAM precharge time and DRAM row access time, which will be tens of ns on a typical DRAM. Thus, the row-buffer hit time could be 30% to 50% less than a row-buffer miss time.

Regarding the efforts of utilizing concurrency among the DRAM banks, one commonly used technique is to interleave small data blocks among memory banks. However, this approach limits the ability to effectively exploit spatial locality in the row-buffer. To consider the trade-offs between large and small data block interleaving schemes, several schemes are proposed. Block interleaving [10] is such an example used in vector supercomputers with Cached DRAM.

In this paper, we analyze the sources of the row-buffer conflicts in the context of superscalar processors. Then we propose a memory interleaving scheme, called *permutation-based page interleaving*, to accomplish both the objectives of utilizing concurrency for reducing row-buffer conflicts and of exploiting access locality for reusing

[1] For Direct Rambus DRAM, the row buffer size is one-half page, and adjacent banks share half-page row buffers with each other.

0-7695-0924-X/2000/$10.00 © 2000 IEEE

the data in the row-buffer. The strategy is to generate the memory bank index by XOR-ing two portions of memory address bits. The hardware cost of the interleaving scheme is trivial, and additional runtime overhead involved is negligible. We evaluate the performance of the proposed interleaving scheme for SPEC95 and TPC-C workloads with execution-driven simulations. Compared with existing schemes, we show that the permutation-based scheme dramatically increases the hit rates on DRAM row-buffers and reduces memory stall time of the workloads. The memory stall times of the workloads are reduced up to 68% and 50%, and the execution times are reduced up to 38% and 19%, compared with the conventional cache line and page interleaving schemes, respectively.

We discuss some issues of memory system design in section 2, and analyze the sources of row-buffer conflicts in section 3. We propose a permutation-based page interleaving scheme in section 4. After introducing our experimental environment in section 5, we present performance comparisons between the permutation-based page interleaving and three other existing schemes in section 6. Other related work is discussed in section 7. Finally, we summarize the work in section 8.

2 Memory System Considerations

2.1 Open-page and Close-page Strategies

An access to DRAM consists of *row access* and *column access*. During row access, a row of data (which is also called a page of data) containing the desired data is loaded into the row buffer. During column access, the data is read or written according to its column address. The page can be either open or closed after an access. Both strategies have their advantages and limitations. In the *open-page* strategy, if the next access to the same bank goes to the same page, only column access is necessary.[2] However, if the next access is a row-buffer miss, the DRAM precharge will not start until the request arrives. The *close-page* strategy allows the precharge to begin immediately after the current access. Which strategy will win mainly depends on the access patterns of applications. If the row-buffer hit rate is high, the open-page strategy should be more beneficial.

Most of our discussions in the rest of the paper are in the context of the open-page strategy. We propose a memory interleaving scheme to improve the row-buffer hit rate. Thus, the open-page strategy is a natural choice for our purpose since it reduces the memory access time for page hits.

2.2 Concurrent Memory Accesses

Most DRAM systems nowadays have multiple banks so that DRAM access operations can be performed on different banks in parallel. Contemporary superscalar processors exploit the instruction-level parallelism (ILP) aggressively by performing out-of-order executions, speculative executions, and non-blocking load/store. A superscalar processor may issue multiple memory requests simultaneously. Although the processor can keep running before the outstanding memory requests are finished, its ability to tolerate long memory latency is still limited [22].

All concurrent memory accesses can be classified into the following three categories:

1. *Accesses to the same page in the same bank.* These accesses fully exploit the spatial locality and can be well pipelined. Precharge and row access are needed to initiate the first access. Subsequent accesses only require column access.
2. *Accesses to different pages in different banks.* Since the accesses can be done in parallel, the corresponding operations can also be well pipelined.
3. *Accesses to different pages in the same bank.* These accesses cause *row-buffer conflicts*. Precharge and row access are needed to initiate each access. The operations cannot be pipelined. Thus, the access patterns belonging to this category have much higher latency than those belonging to the first two categories, and only partially utilize the memory bandwidth.

[2] One cycle is normally required for bus turn-around between read and write accesses.

Parameter	Parameter descriptions
m	the length of the memory address in bits.
Cache-related	**Parameter descriptions**
C	the cache size in bytes.
S	the number of sets in the cache.
N	the number of blocks in a set.
B	the block size in bytes.
s	the length of the cache set index in bits. $s = \log S = \log C/(BN)$.
b	the length of the cache block offset in bits. $b = \log B$.
t	the length of the cache tag in bits. $t = m - (s + b)$.
Memory-related	**Parameter descriptions**
K	the number of memory banks.
P	the page size in bytes, which is also the size of the row_buffer.
R	the number of pages (rows) in a memory bank.
k	the length of the memory bank index in bits. $k = \log K$.
p	the length of the page offset in bits. $p = \log P$.
r	the length of the page index in bits. $r = \log R = m - (k + p)$.

Table 1: Parameters of a memory system.

2.3 Framework of Interleaving Schemes

A memory system is characterized by a group of parameters in Table 1. Figure 1 shows the bit representations of a memory address for conventional cache line and page

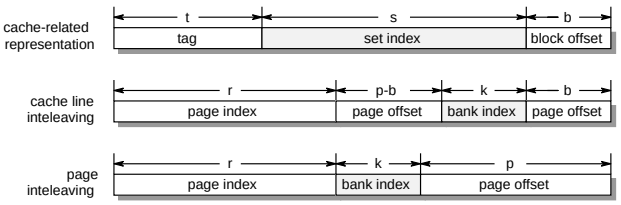

Figure 1: Bit representations of a memory address for both cache addressing and memory addressing with conventional cache line and page interleaving schemes.

interleaving, and gives the relationship between the cache-related representation and the memory-related representation for given memory hierarchical configuration.

The cache line interleaving scheme uses the k bits above the low order b bits (L2 block offset) as the memory bank index. In the uniprocessor system, the processor usually requests data from the memory in a unit of an L2 cache line. The cache line interleaving scheme attempts to access multiple memory banks uniformly (e.g. [5]). However, since continuous cache lines are distributed in different memory banks, this scheme may not effectively exploit the data locality in the row buffer.

The conventional page interleaving scheme uses the k bits above the low order p bits (page offset) as the bank index. This balances between exploiting the data locality in row buffer and referencing memory banks uniformly. However, it may cause severe row buffer conflicts in some typical cases which we will discuss next.

The high order interleaving scheme uses the high order k bits as the bank index. This exploits higher data locality than low order interleaving, but also makes accesses to multiple banks less uniform. In addition, continuous accesses in DRAMs crossing the page boundary will incur precharge and row access. Thus, there is no benefit to exploit spatial locality beyond the page size.

3 Sources of Row-buffer Conflicts

In the conventional page interleaving, there are three major sources for row-buffer conflicts and conflict misses: *L2 cache conflict misses*, *L2 cache writebacks*, and *specific memory access patterns*.

3.1 L2 Conflict Misses

We will use the following example to show that data access patterns causing L2 conflict misses will again cause DRAM row-buffer conflicts and conflict misses under some conditions.

```
double X[T], Y[T], sum;
for (i = 0; i < T; i ++)
   sum += X[i] * Y[i];
```

Without losing generality, we assume the L2 cache is direct mapped, arrays X and Y are contiguously allocated in the memory, and the address distance between $X[0]$ and $Y[0]$ is a multiple of the cache size. Then a pair of data elements $X[i]$ and $Y[i]$ ($i = 0, \cdots, T-1$) will map to the same cache line. Specifically, if a cache line holds E elements, the sequential accesses to $X[0], Y[0], \ldots, X[E-1], Y[E-1]$ will cause L2 conflict misses and generate the following accesses to the main memory:

$$x, y, x, y, \cdots, x, y$$

where x and y are the block addresses of $X[i]$ and $Y[i]$ ($i = 0, \cdots, E-1$), respectively.

What will happen in the DRAM banks for this sequence of memory accesses? To answer this question, we need to look into the bit representations of these addresses. For modern computer systems, the L2 cache size is much larger than the row-buffer (page) size. In addition, the associativity of L2 cache and the number of memory banks are limited. Thus, the bank index is a part of the L2 set index, and the page index comprises the L2 tag (refer to Figure 1).

Since x and y are block addresses mapped to the same cache line, their set indices are the same. Thus, x and y share the same bank index. On the other hand, since x and y are different block addresses, their cache tags must be different. Thus, their page indices are also different. So block addresses x and y are mapped to the same bank but on different pages. In this example, each L2 conflict miss (except the first one) will again cause a DRAM row-buffer conflict miss.

In summary, *any L2 conflicting addresses* (having the same L2 set index but different L2 tags) *are row-buffer conflicting* (having the same bank index but different page indices), providing that the L2 cache size divided by the L2 cache associativity is larger than the accumulated size of all the DRAM row-buffers. For similar reason, in conventional cache line interleaving, any L2 conflicting addresses are also row-buffer conflicting.

3.2 L2 Writebacks

The writeback policy is commonly used in memory systems to preserve data consistency and system efficiency. When an L2 cache miss happens, if the replaced cache block is dirty, it must be written back to the memory or the write buffer before the missed block is read from the memory. Since the read address and the write address belong to different memory locations mapped to the same cache set, they are L2 conflicting addresses. Consequently, they cause a row-buffer conflict under page interleaving. Normally, programs have spatial locality. When a sequence of replacement on dirty cache blocks happens, the reads and writes conflict on the row-buffer and cause frequent row-buffer conflict misses where the pages with the read addresses or the write addresses are replaced and retrieved back and forth.

Write buffers can be used to reduce processor stalls waiting for memory writes [7, 20]. The write buffer can be implemented with read bypass (read misses have higher priority than writes) or with no-bypass. The write buffer with no-bypass will not change the access patterns causing row-buffer conflicts. The write buffer with read bypass can alleviate row buffer conflicts by postponing the writebacks and grouping consecutive reads together. The effectiveness of the write buffer depends not only on its size, but also on when the buffered data are written to the memory. One write policy for reducing the row-buffer conflicts is to write the buffered data to memory only when the number pending writes reaches a threshold. However, since writebacks are not issued immediately when the memory system is free, the delayed writebacks may compete with subsequent reads and increase their latencies. Another write policy is to write the buffered data to main memory whenever there are no outstanding reads. However, the memory access patterns do not change so much in this case. In Section 6, we will show with experiments that using write buffers may reduce row-buffer miss rates but fails to reduce memory stall time effectively.

3.3 Specific Memory Access Patterns

Some specific memory access patterns may cause row-buffer conflicts. For example, when the distance of memory locations between consecutive data elements being accessed is a multiple of the accumulative size of all row buffers of the memory banks, each element is stored in a different page of the same memory bank. Thus, continuous accesses will cause row-buffer conflicts.

4 A Permutation-based Page Interleaving

In order to address the problem of row-buffer conflicts caused by the three sources discussed in the previous section, we introduce a new memory interleaving scheme which generates different bank indices by retaining spatial locality and by reducing row-buffer conflicts. An attractive technique of generating bit patterns used in memory addressing is to XOR the original bit pattern with another bit pattern [14]. Our interleaving scheme is based on this technique.

4.1 The Scheme and its Properties

Our memory interleaving scheme, called *permutation-based page interleaving*, is shown in Figure 2. The low order k bits of the L2 tag and the original bank index are used as the input to a k-bit bitwise XOR logic to generate the new bank index. The page index and the page offset are unchanged. The selection of k bits from the bank index under the conventional page interleaving scheme keeps the same degree of data locality, while the selection of k bits from the L2 tag attempts to make a wide distribution of

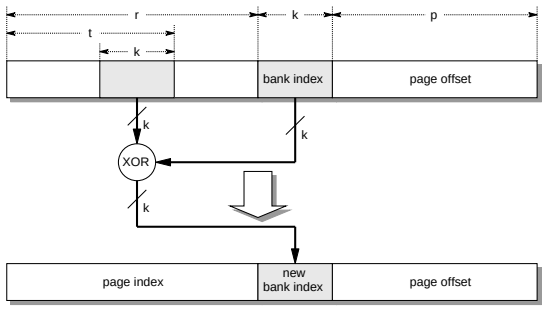

Figure 2: The permutation-based page interleaving scheme

pages among banks for exploiting concurrency. Other design choices could be used with the same mapping principle. We will discuss these later.

Let $\langle a_{m-1}a_{m-2}\cdots a_0\rangle$ be the binary representation of a memory address A. Then the bank index under the conventional page interleaving, I, is $\langle a_{k+p-1}\cdots a_p\rangle$. The new bank index after applying the permutation-based page interleaving scheme, I', is

$$a'_i = a_i \oplus a_{m-t+i-p} \qquad \text{for } i = p, \ldots, k+p-1 \qquad (1)$$

This interleaving scheme has the following properties, which are useful in achieving the objectives of exploiting both the concurrency and the data locality:

1. *L2-conflict addresses are distributed onto different banks.*

   Given any two L2-conflict addresses, their bank indices in conventional page interleaving are identical, but their t-bit L2 tags are different. As long as the low order k bits of the two tags are different, the k-bit XOR function will produce two different bank indices. Figure 3 shows an example of mapping four L2-conflict addresses onto 16 banks. All the four addresses are mapped onto the same bank in conventional page interleaving. After applying the permutation-based page interleaving scheme, they are distributed onto four different banks.

2. *The spatial locality of memory references is preserved.*

   All addresses in the same page are still in the same page after applying our interleaving scheme.

3. *Pages are uniformly mapped onto multiple memory banks.*

   The permutation-based page interleaving scheme still uniformly maps continuous pages onto multiple memory banks, since the conventional bank index information is used in the mapping. Figure 4 gives an example to show that continuous pages are uniformly mapped onto four memory banks by both the conventional and the permutation-based page interleaving schemes.

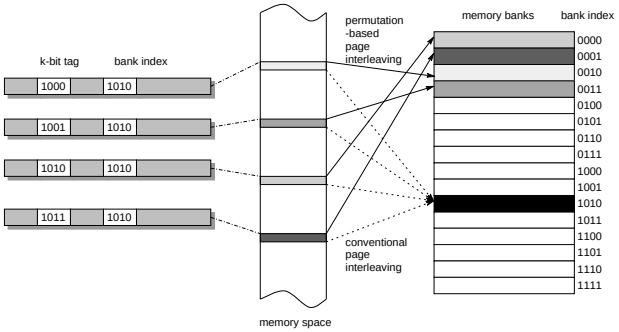

Figure 3: An example of mapping four memory addresses with the conventional page interleaving and the permutation-based page interleaving schemes. Only the k-bit bank index and the low order k-bit of L2 tag are shown for each address.

One would think that spatial locality of memory references could be maintained and page conflicts could be reduced by using only the low order k bits of the L2 tag as the bank index, thus avoiding the XOR operation. The limit of this approach is that it maps a large fraction of the memory space (of the L2 cache size) onto the same bank. This would create hot spots on some memory banks and introduce a new source of page conflicts.

There are several alternatives to the selection of k bits among the t-bit L2 tag. Since programs have data locality, it is more likely that higher order bits of L2-conflict addresses are the same. Our experiments show that choosing the low order k bits achieves or approaches the lowest row-buffer miss rate for all the benchmark programs used.

Other operations such as "add" and "subtract" can also be used to generate the bank index for reducing row-buffer conflicts. However, since this operation is done for each memory access, it should be executed as fast as possible.

We will later show in the paper that the risk for the XOR operation to cause more row-buffer conflicts is very small in practice. A major reason for this is discussed as follows. The memory space can be divided into segments in the unit of the cache size. The XOR operation uses the same k-bit L2 tag for the addresses in each segment. Thus, it does not change the conflicting relationship between any pair of addresses in each segment, which is defined as whether the pair is mapped onto the same row-buffer or not. Our analysis also shows that the XOR operation may increase the chance of conflicts only for addresses in some specific segment boundaries. Since the cache size is sufficiently large in current computer systems, these addresses form a very small subset in the entire memory address space.

4.2 Correctness of the Scheme

The mapping function of a memory interleaving scheme must satisfy the one-to-one property [15]. For a given memory address A, we can obtain its memory location A'

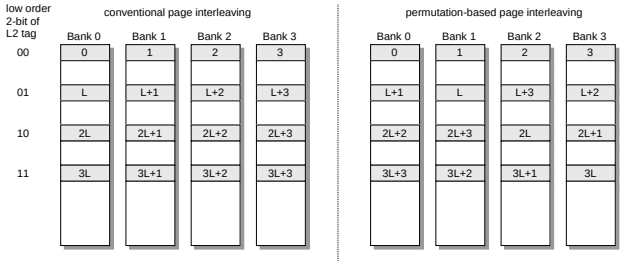

Figure 4: An example of mapping continuous pages onto 4 memory banks under the conventional and the permutation-based page interleaving schemes, where L is the number of pages the L2 cache can hold.

using the permutation-based interleaving scheme by computing its bank index I' using equation (1). Conversely, for a given memory location A', we can obtain its address A by computing $\langle a_{k+p-1}...a_p\rangle$ as $a'_i \oplus a'_{m-t+i-p}$ for $i = p, \ldots, k+p-1$. In modern computer systems, it is always true that $(s+b) > (k+p)$. Thus, for $i = p, \ldots, k+p-1$,

$$a'_i \oplus a'_{m-t+i-p} = (a_i \oplus a_{m-t+i-p}) \oplus a_{m-t+i-p} = a_i. \quad (2)$$

Thus, the permutation-based mapping function has the one-to-one property.

4.3 Comparisons with the Swapping Scheme

Zurawski, Murray, and Lemmon [28] present an interleaving scheme that swaps partial bits of the L2 tag and partial bits of the page offset, which is used in the AlphaStation 600 5-series workstations. We call it the swapping scheme in this paper. Wong and Baer [27] study the performance of the swapping scheme for selected SPEC92 benchmark programs by finding the optimal number of bits to be swapped for these programs.

Figure 5 describes the swapping scheme. This scheme maps every 2^n L2 conflict addresses (with the same $\langle a_{p-1} \ldots a_{p-n}\rangle$) to the same page. Thus, if two L2 conflict misses have the same high order n bits in their page offsets, they will cause page hits. However, if two L2 conflict misses have different high order n bits in their page offsets, they will still cause page conflicts. In addition, the swapping scheme may degrade the spatial locality of memory references because the block size of continuous addresses inside a page is decreased from 2^p to 2^{p-n}. The more bits that are swapped using this method, the more conflict misses can be removed, but the less spatial locality is retained. In contrast, the permutation-based scheme reduces page conflicts and preserves data locality at the same time.

The swapping scheme attempts to convert accesses to different pages in the same bank into accesses to the same page. The permutation-based scheme attempts to convert accesses to different pages in the same bank into accesses

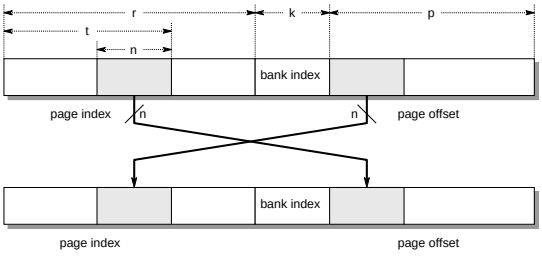

Figure 5: The swapping scheme

to different banks. The permutation-based scheme not only reduces the row-buffer conflicts of current accesses, but also potentially increases the row-buffer hit rates for subsequent accesses.

5 Experimental Environment

Performance evaluation is based on execution-driven simulations. We use the SPEC95 [23] and the TPC-C [24] as workloads, and use SimpleScalar [2] as the base simulator. The database system we have used to support the TPC-C workload is the PostgreSQL (version 6.5) [12].

In order to compare different interleaving schemes, we have modified two programs in the SimpleScalar tool set: *sim-cache* and *sim-outorder*. We use the modified *sim-cache* to measure the row buffer miss rate to compare different interleaving schemes on different memory system configurations at a small simulation cost. This allows us to investigate a wide range of choices. We use *sim-outorder* to measure the execution time and collect detailed statistics of workloads. In addition to the DRAM, the memory controller and a bus with contention are emulated. Bank contention, DRAM precharge, DRAM refresh, and processor/bus synchronization are also considered in the simulation.

We have used *sim-outorder* to configure an 8-way processor, to set the load/store queue size to 32, and to set the register update unit size to 64 in the simulation. The processor allows up to 8 outstanding memory requests, and the memory controller has the ability to accept up to 8 concurrent memory requests. Reads are allowed to bypass writes. The outstanding writes are scheduled to memory modules as soon as there are no outstanding reads. Table 2 gives the major architectural parameters. The 500 MHz processor and the 256-bit (32 bytes), 83 MHz data bus are used in Compaq Workstation XP1000 [4]. All times are converted into processor cycles in the simulation.

6 Performance Evaluation

Using execution-driven simulations with the SPEC95 and TPC-C workloads, we have evaluated the permutation-based page interleaving scheme by comparing it with three other interleaving schemes: cache line interleaving, page interleaving, and swapping.

CPU Clock rate	500 MHz
L1 inst. cache	32 Kbytes, 2-way, 32-byte block
L1 data cache	32 Kbytes, 2-way, 32-byte block
L1 cache hit time	6 ns
L2 cache	2 Mbytes, 2-way, 64-byte block
L2 cache hit time	24 ns
memory bus width	32 bytes
memory bus clock rate	83 MHz
number of memory banks	4∼256
row buffer size	2∼8 Kbytes
DRAM precharge time	36 ns
DRAM row access time	36 ns
DRAM column access time	24 ns

Table 2: Architectural parameters of simulation

6.1 Comparisons of Row-buffer Miss Rates

Figure 6 presents the row buffer miss rates of SPEC95 benchmark programs and the TPC-C workload among the four interleaving schemes: the cache line interleaving (*cache line*), the page interleaving (*page*), the swapping interleaving (*swap*), and our permutation-based page interleaving (*permutation*) schemes. The memory system contains 32 memory banks. The row-buffer size of each bank is 2KB. This is a representative high performance memory system configuration [13].

We have following observations based on our experiments:

- Most programs using cache line interleaving have the highest row buffer miss rates compared with three other interleaving schemes. The row-buffer miss rates of ten benchmark programs out of the total nineteen programs are higher than 90% using cache line interleaving. Since cache line interleaving is normally associated with the close-page mode, the high row-buffer miss rates do not necessarily mean poor overall performance.
- All the programs except *su2cor* using page interleaving have lower miss rates than those using cache line interleaving. However, the miss rate reductions are not significant for most programs.
- Our experiments show that the swapping scheme reduces the row-buffer miss rates for most of the benchmark programs compared with page interleaving. However, the row-buffer miss rates of six programs using the swapping scheme are higher than those using page interleaving. This is because the swapping scheme could make programs exploit less locality than page interleaving, as we have discussed in Section 4.
- For almost all programs, our permutation-based interleaving scheme obtains the lowest row-buffer miss

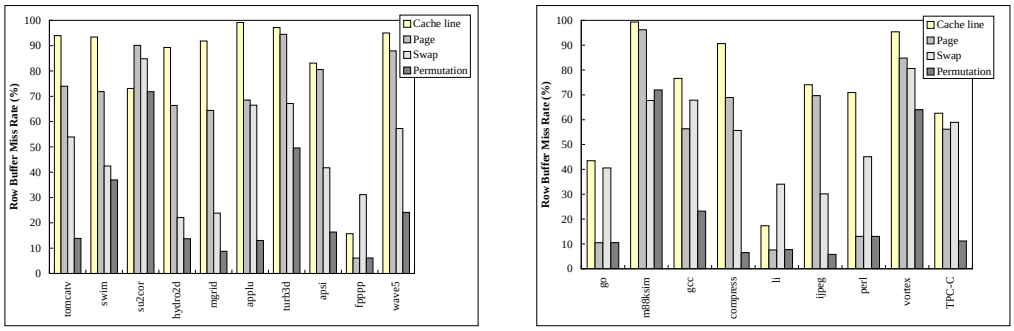

Figure 6: Row buffer miss rates for different interleaving schemes when the number of banks is 32, and the row buffer size is 2KB. The left figure contains SPECfp95 programs, and the right figure contains SPECint95 programs and TPC-C workload.

rates compared with the other three interleaving schemes. The only exception is *m88ksim* whose miss rate is 6% higher than that using the swapping scheme. Our experiments show the permutation-based interleaving scheme significantly reduces the row-buffer miss rates. For example, compared with the best performed interleaving scheme among the other three for each program, the permutation-based interleaving scheme can further reduce the row-buffer miss rate by more than 80% for five programs, and by more than 50% for eight programs.

6.2 Effects of Memory Organization Variations

Changing the number of memory banks and the row-buffer size of each memory bank, we have evaluated the effects of memory system organization variations on the interleaving schemes and on memory performance. Due to the page limit, we only present the performance of selected program *applu*, which is memory intensive and well representative for the group of workloads. Figure 7 shows the row-buffer miss rates of the program using the four interleaving schemes as the number of banks varies from 4 to 256 and the row-buffer size varies from 2 KB to 8 KB.

For each memory system variation, our experiments show that the permutation-based page interleaving scheme reduces the row-buffer miss rate dramatically. For example, when the number of memory banks is 16 and the row-buffer size is 4 KB, the permutation-based interleaving scheme reduces row-buffer miss rates by 82%, 75%, and 72%, compared with the cache-line interleaving, the page interleaving, and the swapping schemes, respectively. We also show that the permutation-based scheme reduces row-buffer miss rate more closely proportional to the increase in the number of memory banks than the other three interleaving schemes. The reason this scheme scales well with the number of memory banks is related to its bank index generation, which is able to widely distribute the conflicted pages among the memory banks. The larger the number of memory banks, the more effective of the permutation-based bank index generation.

6.3 Effects of Write Buffers

Among the nineteen programs we studied, four programs do not have memory write operations. For the rest fifteen programs, the ratios of the number of memory stores to the number of memory loads range from 0.26 to 0.84. Using SPEC95 programs *mgrid* and *applu* as examples, we show the effects of write buffers with different write policies on the row-buffer miss rates. The performance of the other workloads is consistent with that of these two. Figure 8 shows the row-buffer miss rates of *mgrid* and *applu* on a memory system of 32 banks with the row-buffer size of 2KB in each bank. We have compared the following three write policies: *write with no-bypass* (reads are not allowed to bypass writes), *write after reaching the threshold* (writes are scheduled to memory banks only when the number of outstanding writes reaches a threshold — four in our experiments), and *write when memory is idle* (writes are scheduled to memory banks whenever there are no outstanding reads).

As we have discussed in Section 3, postponing writes using write buffers could reduce the row-buffer miss rate. However, our experiments show that the existence of write buffers cannot reduce the row-buffer miss rate as effectively as the permutation-based page interleaving scheme does. For example, when the *write after reaching the threshold* policy is used for program *applu*, the permutation-based scheme can still reduce the row-buffer miss rates by 87%, 65%, and 74%, compared with cache line interleaving, page interleaving, and swapping, respectively.

Although workloads scheduled by the *write after reaching threshold* policy normally get lower row-buffer miss rates than those scheduled by the policy of *write when memory is idle*, the *write after reaching threshold* policy

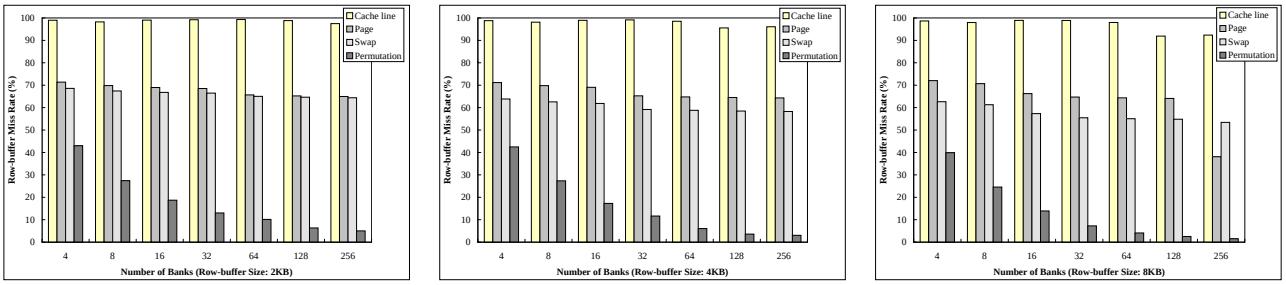

Figure 7: Row buffer miss rates of program *applu* using four interleaving schemes: the cache line, the page, the swapping, and the permutation-based interleaving. The number of memory banks changes from 4 to 256. The performance results in the left figure, the middle one, and the right one correspond to row buffer sizes of 2KB, 4KB, and 8KB, respectively.

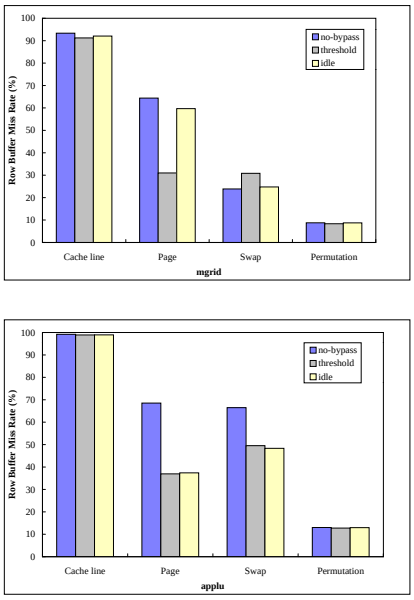

Figure 8: Row buffer miss rates using the three write policies: *write with no-bypass* (no-bypass), *write after reaching the threshold* (threshold), and *write when memory is idle* (idle). The upper figure corresponds to *mgrid*, and the bottom one corresponds to *applu*. The number of memory banks is 32, the row buffer size is 2KB.

may cause higher total execution time due to longer memory stall time. For example, our experiments show that program *mgrid* scheduled by the *write after reaching the threshold* policy reduces the row-buffer miss rate using page interleaving scheme by 48% compared with the policy of *write when memory is idle*, but its total execution time is 12% longer. For this reason, the policy of *write when memory is idle* is used for comparing the overall performance of different interleaving schemes in our study.

6.4 Comparisons of Memory Stall Times

We have measured the memory access portions of CPIs of the SPEC95 programs and the TPC-C workload to compare the four interleaving schemes. In order to show the memory stall portion in each program, we used a method similar to that presented in [1] and [6]. We simulated a system with an infinitely large L2 cache to eliminate all main memory accesses. The difference between the execution time on this "perfect" system and that on a system using the investigated interleaving scheme is defined as the memory stall portion of the program on the system using the interleaving scheme.

We have only studied the SPECfp95 programs and the TPC-C workload because memory accesses only account for a negligible portion in the total execution time for the SPECint95 programs. Figure 9 presents the memory stall portion of the SPECfp95 programs and the TPC-C workload using the four interleaving schemes: the cache line, the page, the swapping, and the permutation-based interleaving schemes. The close-page mode is used for cache line interleaving, while the open-page mode is used for the other three schemes.

Compared with page interleaving, our permutation-based interleaving scheme is able to reduce the memory stall time of these programs by 16% to 50%. The average memory stall time reduction for all the SPECfp95 programs and the TPC-C workload is 37%. Compared with the swapping scheme, our scheme can reduce the memory stall time of these programs by 14% to 53%. The average memory stall time reduction is 33%.

Compared with cache line interleaving, the permutation based interleaving scheme is able to reduce the memory stall time of these programs by 21% to 68%. The only exception is for program *su2cor*, where the memory stall time is increased by 11%. The average memory stall time reduction is 36%. Here is the reason for the exception. Although the permutation-based scheme does reduce the row-buffer miss rate by 8% for *su2cor* compared with the cache line interleaving scheme, the row-buffer miss rate is still as high as 70%. Because cache line interleaving is combined with close-page mode, the precharge can begin

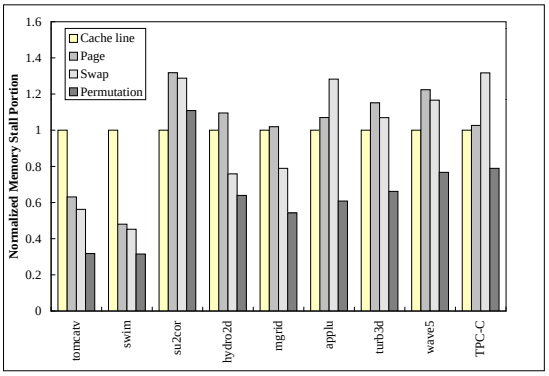

Figure 9: Normalized memory stall portion of the SPECfp95 programs and the TPC-C workload using the four interleaving schemes. All the stall time values are normalized to that using the cache line interleaving scheme. The number of memory banks is 32, and the row buffer size is 2KB.

earlier than in the open-page mode for a row-buffer miss. When the row-buffer miss rate is so high, the benefit of a row-buffer hit cannot offset the penalty caused by late precharge in open-page mode. Thus cache line interleaving outperforms the other schemes which use the open-page mode for this program.

We have also made performance comparisons between cache line interleaving and page interleaving. Among the nine programs we have studied, cache line interleaving outperforms page interleaving for seven programs.

The swapping scheme performs better than cache line interleaving for four programs but worse for five programs. For those four programs, the swapping scheme effectively reduces the row-buffer miss rate so that open-page mode is more beneficial than close-page mode. For most programs, the swapping scheme performs better than page interleaving because the swapping scheme reduces row-buffer conflicts. But for two of these nine programs, the swapping scheme achieves worse performance than page interleaving because data locality cannot be retained after the "swapping".

7 Other Related Work

Hsu and Smith propose and study several memory interleaving schemes which can both increase data locality and avoid hot banks in vector supercomputers with Cached DRAM [10]. There are several other research papers dealing with the bank conflict problem of vector accesses in vector supercomputers. Authors in [8] and [19] attempt to use the prime memory systems to address the conflict issues. Other papers focus on the memory interleaving schemes on vector systems [3, 15, 17, 18, 21, 25]. Authors in [9], [3], and [17] study the skew schemes. Rau, Schlansker, and Yen propose a pseudo-random interleaving technique using the XOR function to randomize the mapping of references to memory modules in [15]. Their scheme can eliminate the occurrence of long clusters due to structured data access. Sohi studies permutation-based interleaving schemes which can improve memory bandwidth for a wide range of access patterns for vector computers [21]. Valero, Lang, and Ayguadé [25] divide the memory address into several portions according to the width of bank index, then XOR all the address portions to generate the bank index. Their method can avoid bank conflict due to power-of-two strides in vector machines. Seznec and Lenfant [18] propose the Interleaved Parallel Scheme, which uses the XOR operation and parameters related to the numbers of processors, logical memory banks, and physical memory banks to induce more equitable distribution over memory banks for a wider set of vectors than the normal mappings.

In contrast to above cited interleaving schemes, our major objective is to reduce the conflicts of DRAM row-buffers. Concurrent accesses to the same bank can be well pipelined in a contemporary DRAM system as long as they hit the row-buffer. In vector machines, concurrent accesses to the same bank always cause bank conflicts and cannot be pipelined.

Besides memory bank interleaving techniques, there are other approaches to address the memory latency problem, such as blocking-free cache, prefetching, thread changing, and data prediction and speculation [26].

8 Conclusion

We have shown that the conflicts and conflict misses of DRAM row-buffers significantly increase memory stall times. We have analyzed their sources in the context of superscalar processors with two levels of caches. Our study indicates that the miss rates of row-buffers are mainly determined by the ways data are interleaved among the memory banks. Conventional schemes, such as cache line and page interleaving, could not effectively exploit both the concurrency of multiple banks and data locality in the row-buffer of each bank. Aiming at achieving the both objectives, we have proposed a memory interleaving scheme, called *permutation-based page interleaving*. By using the fast exclusive-OR operation to generate the bank index, our scheme can dramatically reduce the row buffer miss rates for SPEC95 and TPC-C workloads compared with the two conventional interleaving schemes and an existing optimized commercial scheme. Our execution-driven simulations show that the permutation-based scheme outperforms the cache line interleaving, the page interleaving, and the swapping schemes by reducing the average memory stall times of the workloads by 36%, 37%, and 33%, respectively. In terms of overall performance, the permutation-

based scheme reduces the average execution times of the workloads by 12%, 10%, and 8%, compared with the cache line interleaving, the page interleaving, and the swapping schemes, respectively.

The potential performance penalty of the permutation-based scheme is the exclusive-OR operation for generating each memory bank index. For a modern computer system with multiple levels of caches, this operation is not in the critical path, and can be overlapped with operations above this level in the memory hierarchy. Our experiments show that the additional runtime overhead involved is negligible compared with effective reductions of memory stall times. For example, when using the permutation-based page interleaving scheme, the average memory access latency of the workloads is around 50 CPU cycles, while the exclusive-OR operation only takes about one cycle [11].

Using memory access scheduling techniques to exploit row-buffer locality and concurrency is another attractive approach (e.g. [16]). We believe the combination of access scheduling and the permutation-based interleaving scheme can further improve memory performance.

Acknowledgment:
We appreciate the helpful comments from the anonymous referees. We thank Bill Bynum for reading the paper and for his constructive suggestions. This work is supported in part by the National Science Foundation under grants CCR-9400719 and CCR-9812187, and EIA-9977030, by the Air Force Office of Scientific Research under grant AFOSR-95-1-0215, and by Sun Microsystems under grant EDUE-NAFO-980405.

References

[1] D. Burger, J. R. Goodman, and A. Kägi. Memory bandwidth limitations of future microprocessors. In *Proc. of the 23nd Annual International Symposium on Computer Architecture*, pages 78–89, 1996.

[2] D. C. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report CS-TR-1997-1342, University of Wisconsin, Madison, June 1997.

[3] C.-L. Chen and C.-K. Liao. Analysis of vector access performance on skewed interleaved memory. In *Proc. of the 16th Annual International Symposium on Computer Architecture*, pages 387–394, 1989.

[4] Compaq Computer Corp. *Technology for performance: Compaq professional workstation XP1000*, Jan. 1999. White paper (document number ECG050/0199).

[5] V. Cuppu and B. Jacob. Organizational design trade-offs at the DRAM, memory bus, and memory controller level: Initial results. Technical Report UMD-SCA-TR-1999-2, University of Maryland, Nov. 1999.

[6] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. A performance comparison of contemporary DRAM architectures. In *Proc. of the 26th Annual International Symposium on Computer Architecture*, pages 222–233, May 1999.

[7] J. S. Emer and D. W. Clark. A characterization of processor performance in the VAX-11/780. In *Proc. of the 11th Annual International Symposium on Computer Architecture*, pages 301–310, 1984.

[8] Q. S. Gao. The chinese remainder theorem and the prime memory system. In *Proc. of the 20th Annual International Symposium on Computer Architecture*, pages 337–340, May 1993.

[9] D. T. Harper III and J. R. Jump. Performance evaluation of vector accesses in parallel memories using a skewed storage scheme. In *Proc. of the 13th Annual International Symposium on Computer Architecture*, pages 324–328, 1986.

[10] W.-C. Hsu and J. E. Smith. Performance of cached DRAM organizations in vector supercomputers. In *Proc. of the 20th Annual International Symposium on Computer Architecture*, pages 327–336, May 1993.

[11] W. L. Lynch, G. Lauterbach, and J. I. Chamdani. Low load latency through sum-addressed memory (SAM). In *Proc. of the 25th Annual International Symposium on Computer Architecture*, pages 369–379, 1998.

[12] PostgreSQL Inc. *PostgreSQL 6.5*. http://www.postgresql.org.

[13] Rambus Inc. *256/288-Mbit Direct RDRAM*, 2000. http://www.rambus.com/developer/downloads/rdram_256s_0060_10.pdf.

[14] B. R. Rau. Pseudo-randomly interleaved memory. In *Proc. of the 18th Annual International Symposium on Computer Architecture*, pages 74–83, 1991.

[15] B. R. Rau, M. S. Schlansker, and D. W. L. Yen. The CYDRA 5 stride-insensitive memory system. In *Proc. of the 1989 International Conference on Parallel Processing*, volume 1, pages 242–246, 1989.

[16] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *Proc. of the 27th Annual International Symposium on Computer Architecture*, pages 128–138, 2000.

[17] T. Sakakibara, K. Kitai, T. Isobe, S. Yazawa, T. Tanaka, Y. Inagami, and Y. Tamaki. Scalable parallel memory architecture with a skew scheme. In *Proc. of the 1993 International Conference on Supercomputing*, pages 157–166, 1993.

[18] A. Seznec and J. Lenfant. Interleaved parallel schemes: Improving memory throughput on supercomputers. In *Proc. of the 19th Annual International Symposium on Computer Architecture*, pages 246–255, 1992.

[19] A. Seznec and J. Lenfant. Odd memory systems may be quite interesting. In *Proc. of the 20th Annual International Symposium on Computer Architecture*, pages 341–350, May 1993.

[20] K. Skadron and D. W. Clark. Design issues and tradeoffs for write buffers. In *Proc. of the 3rd International Symposium on High Performance Computer Architecture*, pages 144–155, Feb. 1997.

[21] G. S. Sohi. High-bandwidth interleaved memories for vector processors - a simulation study. Technical Report CS-TR-1988-790, University of Wisconsin - Madison, Sept. 1988.

[22] S. T. Srinivasan and A. R. Lebeck. Load latency tolerance in dynamically scheduled processors. In *Proceedings of the 31st International Symposium on Microarchitecture*, 1998.

[23] Standard Performance Evaluation Corporation. *SPEC CPU95 Version 1.10*, May 1997.

[24] Transaction Processing Performance Council. *TPC Benchmark C Standard Specification, Revision 3.3.3*, Apr. 1998.

[25] M. Valero, T. Lang, and E. Ayguadé. Conflict-free access of vectors with power-of-two strides. In *Proc. of the 1992 International Conference on Supercomputing*, pages 149–156, 1992.

[26] M. V. Wilkes. The memory gap, Keynote Address. In *Workshop on Solving the Memory Wall Problem*, June 2000.

[27] W. Wong and J.-L. Baer. DRAM on-chip caching. Technical Report UW CSE 97-03-04, University of Washington, Feb. 1997.

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