The Journal of Instruction-Level Parallelism

Volume 3, 2002

Breaking Address Mapping Symmetry at Multi-levels of Memory Hierarchy to Reduce DRAM Row-buffer Conflicts

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-buffers have become a critical level of cache in the memory hierarchy to exploit spatial locality in the cache miss stream. Row-buffer conflicts occur when a sequence of requests on different pages goes to the same memory bank, causing higher memory access latency than requests to the same row or to different banks. In this study, we first show that the address mapping symmetry between the cache and DRAM is the inherent source of row-buffer conflicts. Breaking the symmetry to reduce the conflicts and to retain the spatial locality, we propose and evaluate a permutation-based page interleaving scheme. We have also evaluated and compared two representative cache mapping schemes that break the symmetry at the cache level. We show that the proposed page interleaving scheme outperforms all other mapping schemes based on its overall performance and on its implementation simplicity.

1 Introduction

Aiming at reducing the memory access latency, architects have built increasingly deep memory hierarchy to exploit data locality at multi-levels. Besides multi-level caches, the row-buffers of multiple DRAM banks form another level of cache that can be used to exploit the spatial locality in cache miss streams. Researchers have proposed cache mapping schemes to reduce cache conflict misses (e.g. [1, 2, 3, 4]), and DRAM bank interleaving schemes to reduce row-buffer conflicts (e.g. [5, 6]). Instead of focusing on a single level of the memory hierarchy, we examine the inherent source causing conflicts at DRAM row buffers in existing cache and memory mapping structure, and provide effective solutions to reduce the conflicts. This study is built upon our previous work to reduce DRAM row-buffer conflicts [5].

Mathematically, the term of "symmetry" is described as invariance in results under a group of transformations. A simple modular function is the most commonly used for memory address mapping so that only bit selections can be used. For a given memory hierarchy, using different portions of the memory address for cache mapping and DRAM bank interleaving are considered as a group of transformations. We have shown that any L2 conflicting addresses are also row-buffer conflicts under a weak constraint [5]. This finding indicates that address mapping symmetry exists in the conventional address mapping structure that can propagate

conflicts from a higher cache level to the lower levels in the memory hierarchy. In order to break the address mapping symmetry, conventional transformations (cache mapping or memory interleaving scheme) must be changed. The permutation-based page interleaving scheme we proposed in [5] is such an example, which breaks the address mapping symmetry at the DRAM level to reduce row-buffer conflicts. Conducting execution-driven simulations with SPEC2000 benchmark, we provide following new findings and contributions in this study:

- We show that the address mapping symmetry between the cache level and the DRAM level is the architectural source of row-buffer conflicts. Breaking the mapping symmetry can remove this source.

- Examining existing cache mapping schemes and their effects on reducing the conflicts at DRAM row buffers, we show that breaking the address mapping symmetry at the cache level is effective to reduce conflicts at both cache and DRAM row-buffer levels. However, the reduction of average cache miss rates is insignificant, and the increase of processor core complexity by this approach is nontrivial. The results in this study show that our permutation-based page interleaving scheme has the lowest row buffer miss rates and the best overall performance, while the increase of complexity by this approach is trivial and is outside the processor core.

- We evaluate the effects of large cumulative DRAM row-buffer sizes on the effectiveness of the permutation-based page interleaving scheme. We find that the scheme can still be effective for large row-buffers, even when the cumulative row-buffer size is larger than the L2 cache size.

The organization of the paper is as follows. We briefly overview the background of DRAM technology in Section 2. We describe our performance evaluation methodology in Section 3. In Section 4, we provide insights into the address mapping symmetry and show how the conflicts is caused by the symmetry. We present the permutation-based page interleaving scheme, and the performance results in Section 6. We evaluate the effectiveness of several cache mapping schemes on reducing row-buffer conflicts, and discuss their merits and limits

in Section 7. In Section 8, we investigate the case that the cumulative row buffer size is very large. We discuss related work in Section 9, and conclude our study in Section 10.

2 DRAM Memory System Considerations

2.1 DRAM Access Steps

An access to DRAM may consist of three steps of operations: *precharge*, *row access* and *column access*. A DRAM memory system consists of a number of DRAM banks. Every DRAM access uses one of the banks that is determined by the DRAM address mapping. During precharge, the bank is charged to prepare for the following row 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 from row buffer or written to row buffer and DRAM core according to the column address.

Not all the operations are necessary for every access, depending on the state of the bank and the data address to be accessed. A bank can be in *active* or *idle* state. A bank in active state keeps the data in the row buffer valid. If the data to be accessed is in the row buffer, only the column access is necessary (then the memory access is called a *row buffer hit*), otherwise all three operations are required in the order of precharge, row access, and column access (then the memory access is called a *row buffer miss*). A bank changes from active state to idle state after a precharge. The data in the row buffer is lost when the precharge starts. A bank in idle state does not keep the previously accessed data in the row buffer. Row access and column access are required for any access to the bank. The bank returns to the active state after the row access.

The page can be either open or closed after an access finishes. Both strategies have their advantages and disadvantages. In the *open-page* strategy, if the next access to the same bank goes to the same page, only column access is necessary[1]. However, if the next access is a row-buffer miss, the DRAM precharge does not start until the request arrives. The *close-page* strategy allows the precharge to begin immediately after the current access. Which

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

strategy is better depends on the access patterns of applications. If the row-buffer hit rate is high, the open-page strategy should be more beneficial.

2.2 Concurrent Memory Accesses

Contemporary superscalar processors exploit the instruction-level parallelism (ILP) aggressively by performing out-of-order executions, speculative executions, and non-blocking loads. A superscalar processor may issue multiple memory requests simultaneously. Contemporary memory systems can also serve multiple accesses in a pipelined style. In general, concurrent memory accesses have one of the following three patterns:

1. *Accesses to the same page in the same bank.* These accesses fully exploit the spatial locality at the row buffer and can be well pipelined. Precharge and row access are needed to initiate the first access. Subsequent accesses only require column accesses.

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.

In summary, row buffer conflicts affect the DRAM-level concurrency very negatively. Reducing row buffer conflicts not only reduces latency but also improves effective DRAM bandwidth.

2.3 Address Mapping Schemes

Almost all computer systems today use conventional interleaving schemes for both caches and DRAM. Figure 1 shows the bit representations of a memory address for conventional

cache-line and page interleaving, and gives the relationship between the cache-related representation and the memory-related representation for given memory hierarchical configuration. The memory system is characterized by a group of parameters in Table 1.

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.

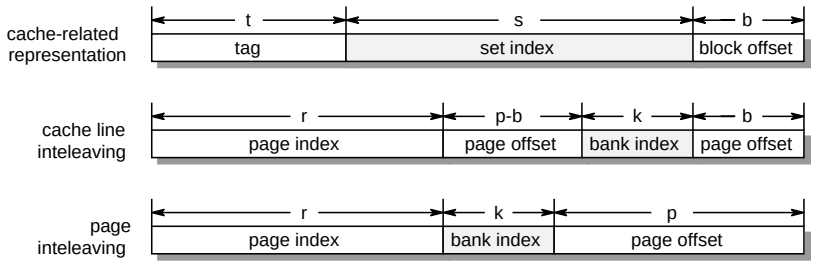

Figure 1: Bit representations of a memory address for both cache addressing and memory addressing with conventional cache-line and page interleaving schemes.

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. [7]). However, since continuous cache lines are distributed in different memory banks, this scheme can 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 Experimental Environment

We use SimpleScalar [8] 3.0b as the base simulation. We inserted simulations of MSHR [9], DRAM, memory controller, contention bus, split bus transaction, and in-order memory access scheduling to the original simulation. Bank contention, DRAM precharge, DRAM refresh, and processor/bus synchronization are also considered in the simulation. We use *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.

We use the SPEC2000 [10] as workloads, which are more memory-intensive than SPEC95. There are thirteen programs with significant memory stall times (measured by the differences

CPU Clock rate	1.6 GHz
L1 inst. cache	32 Kbytes, 2-way, 32-byte block
L1 data cache	32 Kbytes, 2-way, 32-byte block
L1 cache hit time	2 processor cycles
L2 cache	2 Mbytes, 2-way, 64-byte block
L2 cache hit time	10 processor cycles
memory bus width	32 bytes
memory bus frequency rate	133 MHz
number of memory banks	4∼256
row buffer size	1∼8 Kbytes, and 64 KBytes
DRAM precharge time	24 ns
DRAM row access time	24 ns
DRAM column access time	24 ns
L2 MSHR	8 entries
Write buffer	8 entries

Table 2: Architectural parameters of simulation

using two simulations, one with an infinite L2 cache and one with a 2-way 2-MByte L2 cache). We include all those programs in experiments, as shown in Table 3. We use the precompiled SPEC2000 benchmarks provided by Weaver [11](ISA-Alpha). For all programs, we fast-forward 4000M instructions and collect program execution statistics on the next 200M instructions (here 1M = 10^6).

4 Mapping Symmetry and Row Buffer Conflicts

We consider row buffer conflicts in the context of writeback caches with the conventional cache address mapping and DRAM memory with the page-interleaving scheme. We define that two addresses are *cache-conflicting* if they have the same cache index but different cache tags. In other words, they are in different blocks that are mapped to the same cache set. We define that two addresses are *row-buffer-conflicting* if they have the same bank index but different page indices, i.e., they are in different pages of the same bank. We have the following findings:

Program	Description
181.mcf	Combinatorial Optimization
197.parser	Word Processing
168.wupwise	Physics / Quantum Chromodynamics
171.swim	Shallow Water Modeling
172.mgrid	Multi-grid Solver: 3D Potential Field
173.applu	Parabolic / Elliptic Partial Differential Equations
178.galgel	Computational Fluid Dynamics
179.art	Image Recognition / Neural Networks
183.equake	Seismic Wave Propagation Simulation
187.facerec	image Processing: Face Recognition
188.ammp	Computational Chemistry
189.lucas	Number Theory / Primality Testing
301.apsi	Meteorology: Pollutant Distribution

Table 3: Applications selected for experiment that have significant memory stall time. The program *181.mcf* and *197.parser* are integer programs. The others are floating point programs.

- Two cache-conflicting addresses are row-buffer-conflicting, provided the cache size divided by the cache associativity is larger than or equal to the cumulative row buffer size. We call this condition as *large-cache condition*.

  Proof: When the large-cache condition holds, the bits for selecting the bank index is a subset of the bits for selecting the cache set index, and the bits for selecting the page index is a super set of the bits for selecting the cache tag, as shown in Figure 1. Two cache-conflicting addresses have the same cache set index, thus they have the same bank index. On the other hand, they must have different cache tags, so their page indices are different. Therefore, they are row-buffer-conflicting.

- Assume the large-cache condition holds. For writeback caches, the block address of a writeback is row-buffer-conflicting with the block address of the miss that causes the replacement.

  Proof: Writeback happens when a miss causes a replacement on dirty block. The two block addresses must be mapped onto the same cache set, and thus are cache-

conflicting. Thus, they are row-buffer-conflicting.

- Assume the large-cache condition holds. Cache conflict misses may possibly result in row-buffer conflicts. We will use examples to explain this effect in Section 4.3.

Mapping symmetry refers to the fact that both cache and DRAM address mappings use the simple interleaving scheme, and use many common bits for selecting the module to map (cache set at cache level and bank at DRAM level). In particular, when the large-cache condition holds, all bits for selecting DRAM bank are used in the bits for selecting cache set.

4.1 Large-cache Condition in Computer Systems

The large-cache condition is common in today's computers. For a given cache and DRAM chip configuration, there is a threshold of memory size under which the large-cache condition will hold, and this threshold is generally large. For example, assume a computer has a 2MB 2-way associative L2 cache, and its memory system uses DRAM chips that have 8192 rows (pages) per bank[2]. For those chips, the ratio of row buffer size to DRAM capacity is 1:8192. In this example, the large-cache condition holds until the memory size increases beyond 8 GBytes. In practice, it is possible that the memory size is larger than the threshold. However, row buffer conflicts can still be severe. We will discuss this in Section 8.

4.2 Effect of Cache Writebacks

The writeback policy is commonly used for L2 cache on reducing memory bandwidth demand, which has been a crucial issue as the processor speed increases [12]. When a writeback happens, as discussed above, the addresses of the related miss and writeback are row-buffer-conflicting. For writeback and write-allocate cache, either a read miss or a write miss results in a memory read request. The writeback results in a memory write request. Normally, programs have spatial locality. When a sequence of replacement on dirty cache blocks happens, the read requests and the write requests conflict on the row-buffer. This causes frequent row-buffer conflict misses while the pages with the read addresses and the write addresses are replaced and retrieved back and forth.

[2]This is common for 256Mbit SDRAM chips commercially available.

We will use the following example to show this effect:

```
double X[N], Y[N], sum = 0;
int i;
...
for (i = 0; i < N; i ++)
   X[i] = i;
...
for (i = 0; i < N; i ++)
   sum += Y[i];
```

We assume that the cache is direct-mapped, array X and array Y are mapped onto the same cache sets, and array Y is not loaded into cache at the beginning of execution. At the time array Y is accessed, a sequence of misses happens and each miss causes a writeback. From DRAM point of view, a sequence of read requests and a sequence of write requests come to different pages in the same bank during a short time frame when the bank is accessed. In this worst case, each read or write results in a row buffer miss.

Write buffers can be used to reduce processor stalls waiting for memory writes [13, 14]. 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 of 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 6.3, we will show with experiments that using write buffers may reduce row-buffer miss rates but fails to reduce memory stall time.

4.3 The Effect of Cache Conflict Misses

Some typical patterns of cache conflict misses will result in row buffer conflicts. For example,

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

Without losing generality, assume the cache is direct-mapped, the arrays are contiguous in the physical memory space, and $X[0]$ and $Y[0]$ are mapped to the same cache block. Severe cache conflict will happen and each access to $X[i]$ or $Y[i]$ will result in a cache miss. From DRAM point of view, two sequences of read requests come to the same bank interleavingly during a short time frame while the bank is accessed. Each read request will result in a row buffer miss.

Cache conflicts may be reduced by increasing cache associativity, by using victim cache [15], or using other hardware/software approaches. However, this does not alleviate the row-buffer conflicts due to writeback. In those cases, cache conflict misses can be a secondary source of row buffer conflicts.

5 A Permutation-based Page Interleaving

In order to address the problem of row-buffer conflicts caused by cache writebacks and cache conflict misses, we introduce a new memory interleaving scheme which generates different bank indices in a way that retains spatial locality and reduces row-buffer conflicts.

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

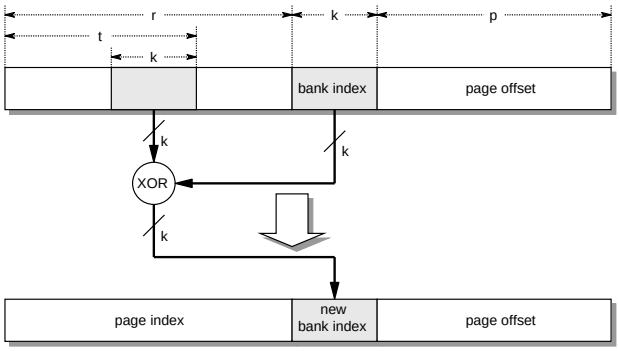

Figure 2: The permutation-based page interleaving scheme

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

$$a'_i = a_i \oplus a_{m-t+i-p} \qquad \text{for } i = p, \ldots, k+p-1 \tag{5.1}$$

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

1. *Cache-conflicting addresses are distributed onto different banks.*

   Given any two cache-conflicting 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.

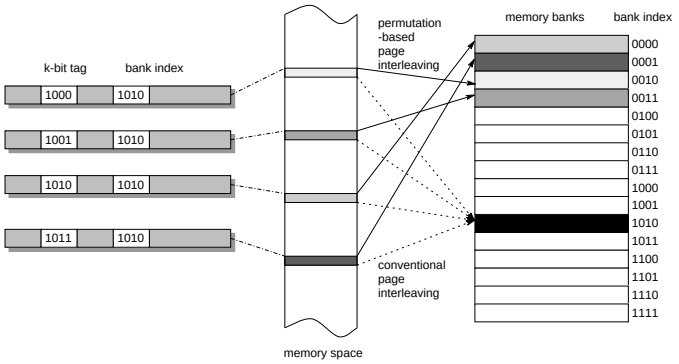

Figure 3: An example of mapping four memory addresses with the conventional page interleaving and the permutation-based page interleaving schemes. Only the k-bit bank index and the low order k-bit of L2 tag are shown for each address.

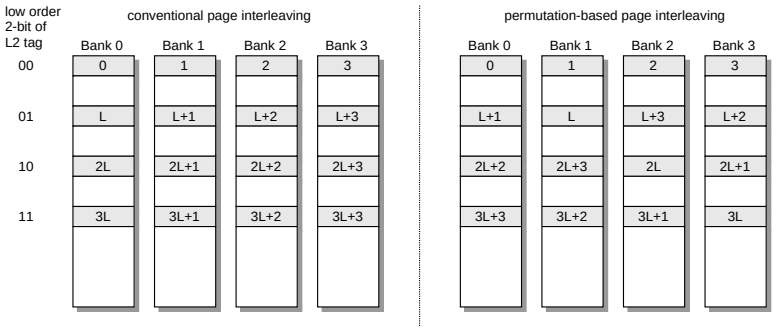

Figure 4: An example of mapping continuous pages onto four memory banks under the conventional and the permutation-based page interleaving schemes, where L is the number of pages the L2 cache can hold.

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.

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.

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.

The mapping function of a memory interleaving scheme must satisfy the one-to-one property [16]. For a given memory address A, we can obtain its memory location A' using the permutation-based interleaving scheme by computing its bank index I' using equation (5.1). 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 $i = p, \ldots, k+p-1$. When the large-cache condition holds, $(s+b) > (k+p)$. Thus, for $i = p, \ldots, k+p-1$,

$$a'_i \oplus a'_{m-t+i-p} = (a_i \oplus a_{m-t+i-p}) \oplus a_{m-t+i-p} = a_i. \tag{5.2}$$

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

5.2 Comparisons with the Swapping Scheme

The swapping scheme is another interleaving scheme that is proposed to reduce the row buffer conflicts. Zurawski, Murray, and Lemmon [17] present the 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 [18] 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 $\langle a_{p-1} \dots a_{p-n} \rangle$) 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.

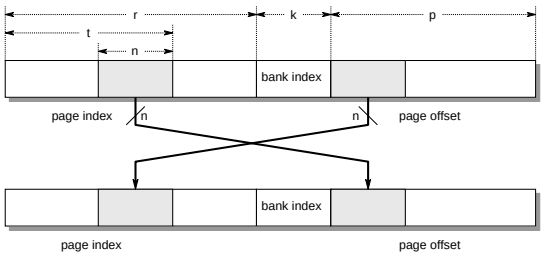

Figure 5: The swapping scheme

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 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.

6 Performance Evaluation of Permutation-based Page Interleaving Scheme

In this section, we evaluate the permutation-based page interleaving scheme by comparing it with three other interleaving schemes: the cache-line interleaving, the page interleaving, and the swapping.

6.1 Reductions of Row-buffer Miss Rates

Figure 6 shows the row buffer miss rates of SPEC2000 programs with the four interleaving schemes: the cache-line interleaving (*cacheline*), the page interleaving (*page*), the swapping interleaving (*swap*), and our permutation-based page interleaving (*page-xor*) schemes. The memory system contains 32 memory banks. The row-buffer size of each bank is 2KB. We use *sim-outorder* in the Simplescalar toolset to collect the row buffer miss rate.

We have following observations:

- All programs using cache-line interleaving have the highest row buffer miss rates compared with the other three interleaving schemes. The average miss rate is 88.7%. Since the cache-line interleaving is normally associated with the close-page mode, its high row-buffer miss rates do not necessarily mean poor overall performance. The other schemes are used with the open-page mode, where the high miss rates do mean poor performance.

- All programs using page interleaving have lower miss rates than those using cache-line interleaving. However, the miss rates are still very high. The average miss rate (arithmetic mean) is 58.6%. Only one program has a miss rate less than 30.0%.

- The swapping scheme may reduce the row-buffer miss rates for some programs but increase the miss rates for others. The average miss rate is 66.3%, higher than that of the page interleaving scheme. The swapping scheme could make programs exploit less locality than page interleaving, as we have discussed in Section 5.

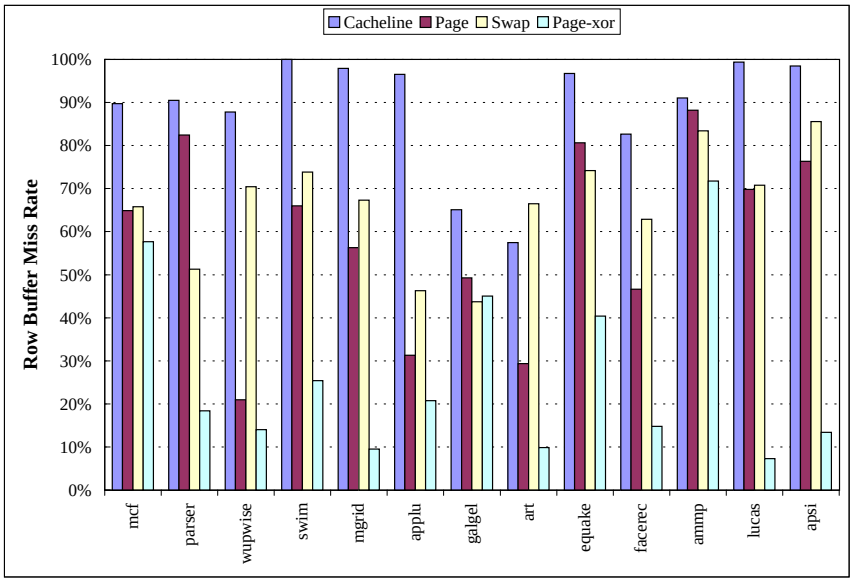

Figure 6: Row buffer miss rates for different interleaving schemes when the number of banks is 32, and the row buffer size is 2KB. The *cacheline* represents cache-line interleaving, the *page* represents conventional page interleaving, the *swap* represents the swap scheme, and the *page-xor* represents the permutation-based page interleaving.

- For almost all programs, our permutation-based interleaving scheme obtains the lowest row-buffer miss rates compared with the other three interleaving schemes. The only exception is *178.galgel* whose miss rate is slightly higher than that using the swapping scheme. The average miss rate is 26.8%. Six programs have miss rates less than 15.0%.

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 space limit, we only present the performance of selected programs *171.swim* and *173.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 1 KBytes to 8 KBytes.

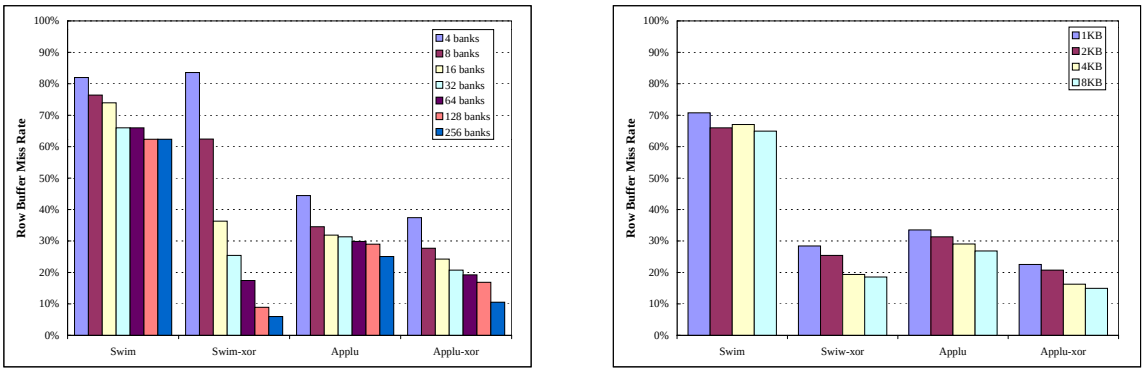

Figure 7: Row buffer miss rates of program *171.swim* and *173.applu* using the conventional page interleaving scheme (without "-xor") and the permutation-based page interleaving (with "-xor"). The left figure shows the effect of changing the number of bank from 4 to 256 with a fixed 2-KByte row buffer size. The right figure shows the effect of changing the row buffer size from 1-KBytes to 8-KBytes with fixed 32 banks.

For each memory system variation, our experiments show that the permutation-based page interleaving scheme reduces the row-buffer miss rate effectively. Furthermore, the permutation-based scheme reduces row-buffer miss rate more closely proportional to the increase in the number of memory banks or the row buffer size than the conventional page interleaving schemes. The reason behind this fact is that the permutation-based bank index generation can 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

For the thirteen programs, the ratios of the number of memory writes to the number of memory reads range from 0.10 to 0.76. Using SPEC2000 programs *172.mgrid* as an example,

we show the effects of write buffer[3] with different write policies on the row-buffer miss rates. The performance of the other workloads is similar. We have compared the following two write policies: *write after reaching threshold* (writes are issued together only when the number of writes reaches a threshold), and *write when memory is idle* (writes are scheduled to memory banks whenever there are no outstanding reads). The later one is what we have used through all other experiments.

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 may cause higher total execution time due to longer memory stall time. For example, our experiments show that program *172.mgrid* scheduled by the *write after reaching the threshold* policy has a 23% row-buffer miss rate with page interleaving scheme, compared with a 56% row-buffer miss rate using the policy of *write when memory is idle*, however, the CPI is increased from 0.68 to 0.92. This is because buffered write requests will stall read requests when those requests are issued together, and in turn they stall the processor. 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.

A major function of the write buffer is to allow memory reads bypass memory writes so that write requests will not stall the processor. To improve the bus utilization, write requests should be issued as long as the bus is idle and there is no pending read. To avoid row buffer conflicts, however, write requests should be held until no future reads will access the same pages. To design a scheduling policy to meet the two conflicting goals is difficult, and may significantly increase the size requirement for the write buffer. In contrast, using our interleaving scheme to avoid such conflicts is much simpler.

6.4 Overall Performance Improvement

Figure 8 gives the CPI of twelve SPEC2000 programs (excluding *181.mcf*) using the four schemes. We exclude *181.mcf* because its CPI values are much higher than other programs which would distort other bars. The close-page mode is used for cache line interleaving, while the open-page mode is used for the other three schemes. We also show the CPI of

[3]This write buffer is located between the L2 cache and the main memory and is used to hold writebacks.

a base system, which is a system with an infinitely large L2 cache to eliminate all main memory accesses. The CPI of the base system provides a lower bound for any performance improvement on DRAM memory systems. We use CPI instead of IPC so as to show how much the permutation-based mapping reduces the memory stall time, which is represented by the difference between the CPI of the base scheme and those of other schemes. We will use the harmonic means of IPC to compare the average performance.

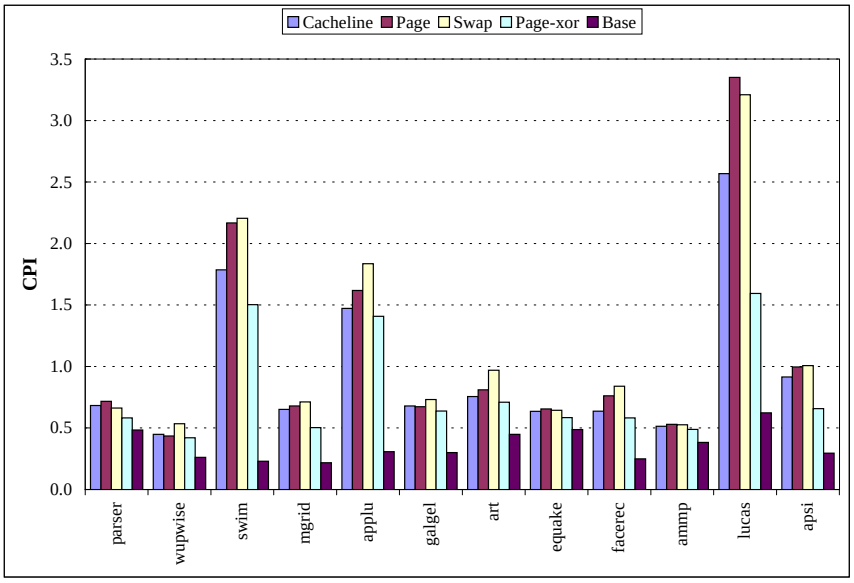

Figure 8: CPI of the twelve SPEC2000 programs using the four interleaving schemes. The number of memory banks is 32, and the row buffer size is 2KB.

Among the three mapping schemes except our permutation-based page interleaving, the average performance of cache-line interleaving is better than the other two. This is because it uses the close-page mode, and because the row buffer miss rates for the other two schemes are very high. The permutation-based page interleaving is better than the cache-line interleaving on all programs except *181.mcf*, which is not shown in Figure 8. The CPI values of *181.mcf* are 7.3 and 7.0 for the permutation-based mapping and the cache-line interleaving,

respectively. This program is bandwidth-bounded. Except *181.mcf*, the permutation-based scheme outperforms all other schemes on all programs. The harmonic mean of IPC for the cache-line interleaving, the page interleaving, the swap scheme, and the permutation-based interleaving are 0.70, 0.61, 0.57, 0.77, respectively (including *181.mcf*). Using this metric, the average improvement of the permutation-based scheme over the cache-line scheme is 11%.

7 Breaking Mapping Symmetry at Cache Level

Researchers have studied cache mapping schemes to reduce cache conflict misses. Two representative schemes are bitwise-XOR [2] and polynomial mapping [19, 2, 3]. Those cache mapping schemes also break the address mapping symmetry but at the cache level. Thus, they may also reduce the row buffer conflicts. For this purpose, the effectiveness of those schemes is determined by how successfully they reduce the possibility that two cache-conflicting addresses are row-buffer-conflicting.

In this section, we examine cache mapping schemes aiming at reducing row buffer miss rates, and discuss the tradeoffs between using cache mapping schemes and using DRAM interleaving schemes.

7.1 Bitwise-XOR and Polynomial Mapping

In the bitwise-XOR scheme, the least significant s bits of the tag are XORed with s set index bits to form the new cache set index, where s is the number of bits in cache set index. The polynomial mapping scheme [19] uses equation $R(x) = A(x) \bmod P(x)$ to map a given address onto a module (here a cache set), where $R(x)$, $A(x)$, $P(x)$ are polynomials over Galois Field GF(2). In the equation, $A(x)$ is the polynomial associated with the address to be mapped, $R(x)$ is the polynomial associated with the cache set index, and $P(x)$ is an irreducible polynomial of order s. The polynomial mapping is effective in avoiding conflicts for strided access patterns. It has been proven that any sub-sequence of length M within strides of form 2^k will be evenly mapped onto M module, which k is a positive integer. The bitwise-XOR scheme can be implemented using single-level XOR gates with two inputs. The

polynomial mapping can be implemented using single-level XOR gates with multiple inputs.

7.2 Reduction of Miss Rates

We first compare the cache miss rates of the two cache mapping schemes with the conventional cache mapping. We use those schemes only for L2 caches but not for L1 caches. For the polynomial mapping, we choose arbitrarily the polynomial $P(x)$ associated with prime number 1572821. Table 4 shows the L2 cache miss rates for the thirteen SPEC2000 programs. The *cache-xor2* represents a revised bitwise-XOR that we will discuss soon in this section.

The miss rates of the two cache mapping schemes are almost identical with the conventional mapping schemes except for program *178.galgel*, *179.art*, *188.ammp*, and *189.lucas*. For *178.galgel*, the polynomial mapping scheme reduces the cache miss rate dramatically, but the bitwise-XOR does not. For *179.art*, both schemes increase the cache miss rates by almost two times. For *188.ammp*, both schemes reduce the miss rates and the bitwise-XOR does better. For *189.lucas*, the bitwise-XOR increases the miss rate by almost 35%, but the polynomial mapping reduces the miss rate by more than 20%. On average, the polynomial mapping reduces the miss rate from 23.5% to 22.0%, and the bitwise-XOR increases the miss rate to 25.0%. The increase or decrease of the average miss rate is not significant, which confirms the previous studies.

We show the row-buffer miss rates of the two cache mapping schemes in Figure 9. We also include the row-buffer miss rates of the permutation-based DRAM page interleaving for comparison, where the conventional cache mapping is used. The bitwise-XOR scheme has the highest row buffer miss rates for all programs. If we compare it with the conventional DRAM mapping (in Figure 6), we will find this scheme only moderately reduces the row buffer miss rate. The row buffer miss rates using the polynomial mapping are close to those of the permutation-based DRAM page interleaving. However, the later one is still better for all applications. The average row buffer miss rates are 47.6%, 34.4%, and 26.8% for the bitwise-XOR cache mapping, the polynomial cache mapping, and the permutation-based DRAM mapping, respectively.

Here is our explanation on why the bitwise-XOR scheme results in high row buffer miss

Programs	Default	Cache-xor	Cache-poly	Cache-xor2
mcf	44.1%	43.2%	44.1%	44.0%
parser	8.4%	8.5%	8.6%	8.4%
wupwise	39.7%	39.7%	39.8%	39.7%
swim	27.9%	27.9%	28.0%	27.9%
mgrid	22.8%	22.8%	22.9%	22.8%
applu	37.2%	37.2%	37.1%	37.2%
galgel	18.2%	18.3%	1.7%	1.4%
art	3.2%	11.8%	11.5%	12.0%
equake	10.6%	10.6%	10.8%	10.6%
facerec	22.7%	22.3%	23.0%	22.8%
ammp	6.5%	2.9%	3.9%	7.0%
lucas	43.5%	58.6%	33.3%	33.3%
apsi	21.2%	21.2%	21.2%	21.2%
Average	23.5%	24.0%	22.0%	22.2%

Table 4: L2 cache miss rates for the conventional cache mapping (default), the bitwise-XOR (cache-xor), the polynomial mapping (cache-poly), and the revised bitwise-XOR (cache-xor2).

rates. In this scheme, the k tag bits that are XORed with the k bank index bits are not the least significant k bits in the tag. In the program memory space, cache-conflicting addresses that differ only in the least significant k bits have a shorter distance than other cache-conflicting addresses. Thus, because of program locality, the least significant k bits change more frequently than other bits in the access stream generated by a program. Under this scheme, the least significant k bits are XORed with bits for selecting DRAM page offset, and the k bits XORed with the bank index changes less frequently. Consequently, from the DRAM point of view, two cache-conflicting addresses appearing within a short time frame are likely to have the same bank index, causing them conflicts at the row buffer.

To confirm this, we switch the two portions of the tag bits such that the least significant k bits are XORed with the bits for selecting the bank index. This revised bitwise-XOR is labeled as *cache-xor2*. The new scheme significantly reduces the row buffer miss rates, and is slightly better than polynomial cache mapping. The average row buffer miss rate is 30.0%. The cache miss rates of this scheme are also shown in Table 4.

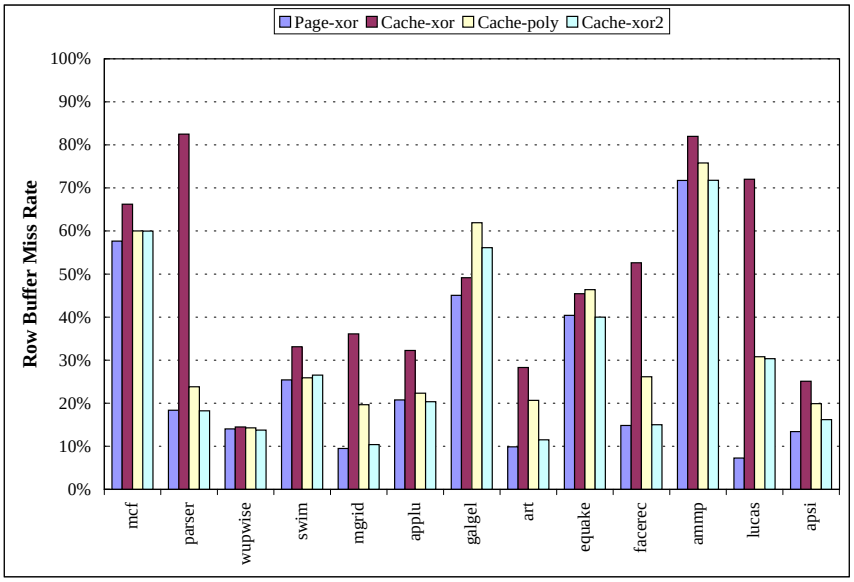

Figure 9: Row-buffer miss rates for the permutation-based mapping (page-xor), the bitwise-XOR mapping (cache-xor), the polynomial mapping (cache-poly), and the revised bitwise-XOR (cache-xor2).

7.3 Comparisons of Overall Performance

Figure 10 shows the CPI of the programs (again *181.mcf* is excluded). For the cache mapping schemes, we do not consider in the simulation the possible delay of critical path by using the mapping. The bitwise-XOR cache mapping has the worst performance for most programs because of the severe row buffer conflicts and the slightly higher average cache miss rate compared with other schemes. Its performance for *189.lucas* is extremely worse than the other schemes because the other schemes have both lower cache miss rate and lower row buffer miss rate. The permutation-based DRAM mapping has the best performance for most programs because it has the lowest row buffer miss rates. It is much better than the others for *179.art* because the other three schemes increase the cache miss rate. However, it is worse than the polynomial mapping and the revised bitwise-XOR for *178.galgel* because the

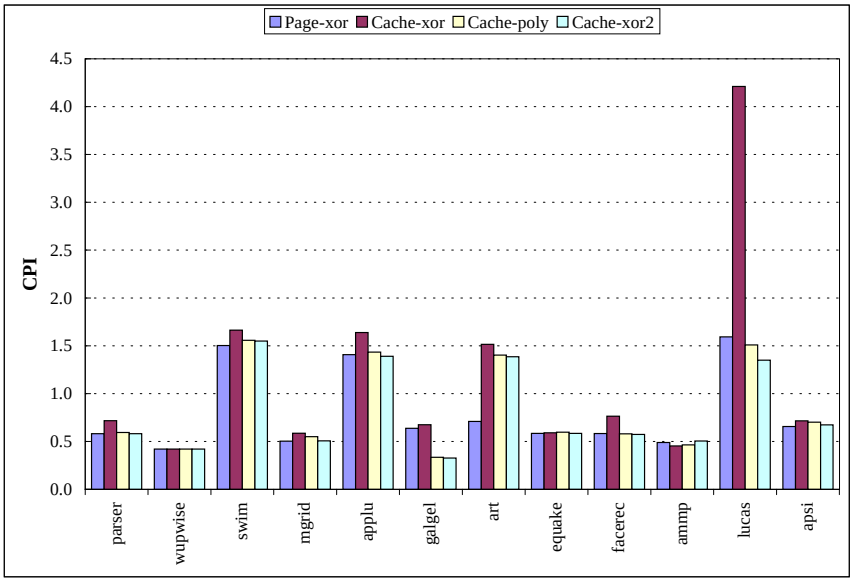

Figure 10: CPI of twelve SPEC2000 programs using the permutation-based DRAM mapping (*page-xor*)), the bitwise-XOR cache mapping (*cache-xor*), the polynomial cache mapping (*cache-poly*), and the revised bitwise-XOR cache mapping (*cache-xor2*).

two cache mapping schemes reduce the cache miss rate dramatically. When being successful in reducing cache miss rates, the polynomial cache mapping and the revised bitwise-XOR cache mapping perform better than the permutation-based DRAM mapping. Otherwise, the permutation-based DRAM mapping performs slightly better than the revised bitwise-XOR cache mapping, and the later one performs slightly better than the polynomial cache mapping, because of the difference in row buffer miss rates.

The harmonic means of IPC are 0.77, 0.60, 0,74, and 0.75 for the permutation-based DRAM mapping, the bitwise-XOR cache mapping, the polynomial cache mapping, and the revised bitwise-XOR cache mapping, respectively. The result indicates that the advantage of the permutation-based DRAM mapping on reducing the row buffer miss rate is so effective that its disadvantage of no consideration of cache miss reduction becomes insignificant.

7.4 Tradeoffs between Cache Mapping Schemes and DRAM Interleaving Schemes

As discussed in the previous subsection, using different cache mapping schemes (for example, the polynomial one or the revised bitwise-XOR) may significantly reduce the row buffer miss rate. However, our performance results have shown that the average performance improvement from reducing cache conflict misses is insignificant. Compared with conventional cache mapping and DRAM interleaving, almost the entire overall performance gain comes from the reduction of row buffer conflict misses. Nevertheless, using a polynomial cache mapping scheme may have the advantage of good predictability of cache behavior [3]. If the predictability of cache behavior is important, cache mapping schemes like the polynomial mapping are attractive because they can reduce conflicts at both cache and row buffer levels.

However, the implementations of these cache mapping schemes are nontrivial [3]. They should not increase the delay in the critical path. When multiple-level caches maintain the property of inclusion, i.e., the data cached at a higher level must be cached at the lower level, it is necessary to enforce explicit invalidation in the higher level cache when a cache block in the lower level cache is replaced. Although those issues are addressable, the solutions do increase the complexity of the processor core. In comparison, the permutation-based page interleaving scheme does not have such implementation concerns, and the logic is implemented outside the processor core. In short, the scheme is much more cost-effective by considering both the significant performance gain and its simplicity.

8 Considerations of Large Cumulative Row Buffer Sizes

All of our analyses on row buffer conflicts so far have been based on the large-cache condition, which is normally realistic. However, if the memory size is very large, the cumulative row buffer size may be larger than the cache size divided by the cache associativity. In this section, we examine a memory size threshold for the large-cache condition to hold, and investigate how the increase of memory size beyond the threshold will affect the effectiveness of the permutation-based scheme.

Assume W is the value of the cache size divided by the cache associativity. In a DRAM

memory system, the ratio of DRAM capacity to the row buffer size is usually a constant R for all DRAM chips. Thus, the ratio of the memory size, m, to the row buffer size is R. The product of W and R, denote as M, is a threshold for m. The large-cache condition holds when and only when $m \leq M$. R is large in practice, for example, 8192 for today's 256Mbit SDRAM chips. In other words, the threshold M is 8 GBytes with a 2-way set associative, 2-MByte L2 cache and a DRAM system with such chips.

When $m > M$, we are specially interested in the cases that m is a small multiple of M, for example, $2M$, $4M$, or $8M$, for practical reason. As m increases, eventually the row buffer miss rate will be close to zero even under the conventional DRAM mapping. However, this requires a very large memory size. The advantage of the permutation-based scheme is still obvious when m/M is small. Under the conventional DRAM mapping, any two cache-conflicting addresses can now be distributed to m/M row buffers instead of one row buffer (assume m/M is less than the number of row buffers). Thus, the increase of m/M will reduce the row buffer miss rate. However, the permutation-based scheme can distribute those addresses onto all row buffers, whose number can be much larger than m/M in practice.

When $m > M$, the tag bits and the k bits of bank index are partially overlapped. We slightly change the permutation-based scheme as follows: instead of using the least significant k bits in the tag for XORing, we use the least significant k bits in the tag portion that are not overlapped with the bank index. This is necessary to guarantee the correctness of the scheme because XORing overlapping bits will make the scheme lose the one-to-one property.

Figure 11 shows the row buffer miss rates for all the thirteen programs for different numbers of banks and a fixed row buffer size, each with the conventional page interleaving and with the permutation-based page interleaving. The number of banks is 32, 64, or 128, and the row buffer size is 64KB. The W here is 1MB, and the cumulative row buffer size is 2, 4, and 8 times of W, respectively. With $r = 8192$, the memory size threshold M is 16 GBytes, 32 GBytes, and 64 GBytes respectively. With the conventional page interleaving, the average row-buffer miss rates are 30.0%, 14.1%, and 17.1%, respectively. With the permutation-based page interleaving, the average row-buffer miss rates are 14.5%, 8.1%, and 5.1%, respectively. In summary, using the permutation-based page interleaving is still

effective to improve the application performance even when the memory size is beyond the threshold.

Using memory access scheduling techniques to exploit row-buffer locality and concurrency is another attractive approach (e.g. [20]). We believe the combination of access scheduling and the permutation-based interleaving scheme can further improve memory performance.

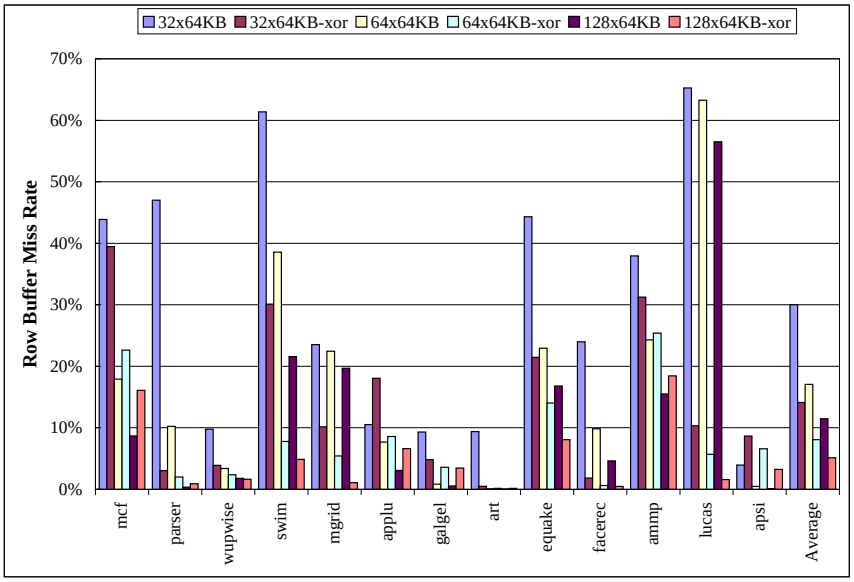

Figure 11: The row buffer miss rates of conventional and permutation-based page interleaving schemes when the cumulative row buffer size is larger than cache size divided by cache associativity. In the legend, 32, 64, or 128 before the "x" represents the number of bank, 64KB represents the size of the row buffer, and the "xor" indicates using the permutation-based page interleaving.

9 Other Related Work

Hsu and Smith propose and evaluate several memory interleaving schemes that can both increase data locality and avoid generating hot banks in vector supercomputers with cached

DRAM [21], where processors do not have data caches. Our study targets on superscalar processors with DRAM memory systems. The large caches in our targeted systems make the memory access patterns significantly different from that in the vector system without caches.

There are several other research papers dealing with the bank conflict problem of vector accesses in vector supercomputers. Authors in [22] and [1] attempt to use the prime memory systems to address the conflict issues. Other papers focus on the memory interleaving schemes on vector systems [23, 16, 24, 25, 26, 27]. Authors in [28], [23], and [24] 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 [16]. 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 [26]. Valero, Lang, and Ayguadé [27] 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 [25] 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.

The above cited studies are based on vector supercomputers with SRAM memory systems. Besides different memory access patterns on those machines, the sources of access conflicts in our targeted systems are also different from those in the vector machines without DRAM memory systems. For example, elimination of DRAM row buffer conflicts without reducing the available locality is a major issue in our study. Therefore, our study has a different objective with a different focus.

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.

10 Conclusion

We have shown that the address mapping symmetry is the inherent source of row buffer conflicts under conventional cache and DARM address mapping. Breaking the mapping symmetry, the proposed permutation-based page interleaving scheme can eliminate or significantly reduce the severe row buffer conflicts and retain the spatial locality. Conventional schemes, such as cache-line and page interleaving, could not effectively exploit both the DRAM concurrency and spatial locality in the row-buffer. Our execution-driven simulations show that the permutation-based scheme can significantly reduce the row buffer miss rates and improve the overall performance.

We have also shown that the mapping symmetry can be broken at the cache-level to remove this source of row buffer conflicts. We have evaluated two representative cache mapping schemes, bitwise-XOR and polynomial mapping, which are proposed originally for avoiding cache conflict misses. The polynomial mapping can reduce the row buffer miss rate close to that of the permutation-based page interleaving, but the bitwise-XOR must be modified to avoid conflicts. Our results indicate that, conflict-avoiding cache mapping schemes should also consider the conflicts at the row buffer. We show that almost all performance gains come from the reductions of row buffer miss rates, and the permutation-based page interleaving scheme has the best overall performance. Considering the scheme does not increase the complexity of processor core, it is also the most cost-effective approach.

In Table 5, we give a summary of the three cache mapping and memory interleaving schemes, namely, the bitwise-XOR cache mapping, the polynomial cache mapping, and our permutation-based page interleaving scheme. We present their impacts on three aspects of performance, namely, cache conflict reduction, row-buffer conflict reduction, and overall performance improvement, and their impacts on increasing implementation complexity. Our study shows that the permutation-based page interleaving scheme outperforms all other schemes based on its performance improvement and on its implementation simplicity.

Acknowledgment: This research project has been motivated by many discussions with our colleagues in the field. Antonio González of Universitat Politcnica de Catalunya provided us with additional references on cache mapping techniques. Comments from Allen Baum of Compaq Computers are helpful for us to further look into the effects of L2 cache associativity.

Mapping Scheme	Cache conflict reduction	Row-buffer conflict reduction	Overall performance improvement	implementation complexity
Bitwise-XOR cache mapping	moderate	low	low	moderate
Polynomial cache mapping	moderate	high	high	high
Permutation-based DRAM page interleaving	N/A	highest	highest	low

Table 5: Summary of the three cache mapping and memory interleaving schemes, and their impact on three aspects of performance, and on implementation complexity.

Discussions with Kevin Normoyle of Sun Microsystems provide us with industries' perspectives on memory systems. Our work is also beneficial from communications with Zhiyong Liu of Chinese Academy of Sciences, who did analytical work on cache mapping. We are grateful to our colleague Bill Bynum for reading the paper and his constructive comments.

This work is supported in part by the National Science Foundation under grants CCR-9812187, EIA-9977030, and CCR-0098055. In addition, this work is also a part of an independent research project sponsored by the National Science Foundation for its program directors and visiting scientists.

References

[1] A. Seznec, "A case for two-way skewed-associative caches," in *Proceedings of the 20th Annual International Symposium on Computer Architecture*, May 1993, pp. 169–178.

[2] A. Gonzalez, M. Valero, N. Topham, and J. M. Parcerisa, "Eliminating cache conflict misses through XOR-based placement functions," in *Proceedings of the 11th Inter-*

national Conference on Supercomputing, New York, July 7–11 1997, pp. 76–83, ACM Press.

[3] F. J. Sánchez and A. González, "Cache sensitive modulo scheduling," in *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture*, Los Alamitos, Dec. 1–3 1997, pp. 338–348, IEEE Computer Society.

[4] Z. Liu and X. Li, "XOR storage scheme for frequently used data patterns," *Journal of Parallel and Distributed Computing*, vol. 25, no. 2, pp. 162–173, Mar. 1995.

[5] Z. Zhang, Z. Zhu, and X. Zhang, "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," in *Proceedings of the 33rdIEEE/ACM International Symposium on Microarchitecture*, dec 2000, pp. 32–41.

[6] W. Lin, S. Reinhardt, and D. Burger, "Reducing dram latencies with an integrated memory hierarchy design," in *Proceedings of the Seventh International Symposium on High Performance Computer Architecture*, Jan. 2001.

[7] 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*, May 1999, pp. 222–233.

[8] 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.

[9] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," in *Proceedings of the 8th Annual International Symposium on Computer Architecture*, May 1981, pp. 81–87.

[10] J. L. Henning, "SPEC CPU2000: measuring CPU performance in the new millennium," *IEEE Computer*, vol. 33, no. 7, pp. 28–35, July 2000.

[11] C. Weaver, *http://www.simplescalar.org/spec2000.html*, SPEC2000 binaries.

[12] 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*, 1996, pp. 78–89.

[13] 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*, 1984, pp. 301–310.

[14] 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*, Feb. 1997, pp. 144–155.

[15] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *Proceedings of the 17th Annual International Symposium on Computer Architecture*, May 1990, pp. 364–373.

[16] 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*, 1989, vol. 1, pp. 242–246.

[17] J. H. Zurawski, J. E. Murray, and P. J. Lemmon, "The design and verification of the AlphaStation 600 5-series workstation," *Digital Technical Journal*, vol. 7, no. 1, pp. 89–99, 1995.

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

[19] B. R. Rau, "Pseudo-randomly interleaved memory," in *Proceedings of the 18th Annual International Symposium on Computer Architecture*, 1991, pp. 74–83.

[20] 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*, 2000, pp. 128–138.

[21] 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*, May 1993, pp. 327–336.

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

[23] 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*, 1989, pp. 387–394.

[24] 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*, 1993, pp. 157–166.

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

[26] 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.

[27] 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*, 1992, pp. 149–156.

[28] 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*, 1986, pp. 324–328.