Exploiting Program Cyclic Behavior to Reduce Memory Latency in Embedded Processors

Ehsan Atoofian
Electrical and Computer Engineering Department
University of Victoria
Victoria BC, Canada
eatoofia@ece.uvic.ca

Amirali Baniasadi
Electrical and Computer Engineering Department
University of Victoria
Victoria BC, Canada
amirali@ece.uvic.ca

ABSTRACT

In this work we modify the conventional row buffer allocation mechanism used in DDR2 SDRAM banks to improve average memory latency and overall processor performance. Our method assigns row buffers to different banks dynamically and by taking into account program cyclic behavior and bank row buffer demand.

As we show in this work, memory requests go through several phases. In each phase, programs tend to access a single bank most of the time. We exploit this repetitive behavior and improve the concurrency level for memory read and write operations. We do so by assigning idle row buffers to more demanding banks during specific program phases. This improves average memory latency and processor performance by 12.7% and 7.6% respectively.

Categories and Subject Descriptors

C.1.1 [**PROCESSOR ARCHITECTURES**]: Single Data Stream Architectures.

General Terms

Design, Measurement, Performance.

Keywords

Memory, Row Buffer, High-Speed Embedded Processors.

1. INTRODUCTION

The speed of embedded processors has improved significantly largely due to innovative techniques introduced at architectural and VLSI levels. Despite such advances, the main memory continues to serve as a bottleneck. This is the result of the fact that processor speed has improved at a much faster pace compared to memory. This increasing speed gap between memory and processor has made memory design a critical issue in embedded processors. While exploiting caches has been able to bridge the gap between processor and main memory performance, as some studies have suggested, caching is becoming less effective as the gap grows [1]. The growing impact of memory on the overall system performance motivates designers to seek performance optimization techniques for the memory system.

One possible solution to this problem is to overlap memory accesses, servicing several memory requests in parallel. To increase the number of memory requests executed in parallel (also referred to as memory level parallelism) modern memory structures allow pipelining memory requests. This is facilitated by providing several independent banks, and caching the most recently accessed row in each bank. While these features improve memory bandwidth, the overall improvement would be highly dependent on memory requests patterns. Improvements are at their peak when the same bank row is accessed continuously or different banks are accessed sequentially. Unfortunately, real applications do not always follow such a pattern. For example, patterns including several consecutive accesses to different rows of a bank incur row conflict, effectively stalling memory access pipelining.

This work aims at reducing memory latency by dynamically allocating memory resources (i.e., row buffers) to memory banks using memory accesses cyclic behavior. We refer to our technique as row buffer sharing (RBS).

RBS aims at reducing latency in DDR2 SDRAMs. DDR2 SDRAMs are composed of several banks, each equipped with one row buffer. The row buffers act as a cache and hold the most recent accessed row of the banks. Having multiple banks increases parallelism among memory requests and reduces memory response time.

Our study shows that memory requests often access only a single bank consecutively and for long periods. During such periods, other banks are idle and their row buffers remain unused. In RBS we assign such idle row buffers to the active bank and increase the number of rows simultaneously cached in the row buffers. As such, we reduce memory latency for a larger number of memory accesses. Also, faster memory accesses improves processor overall performance.

To achieve high performance improvements, it is important that the number of active memory rows stay relatively low and do not exceed the number of available buffers. Otherwise RBS cannot improve memory latency considerably as row buffers will be replaced continuously as new memory requests arrive. Fortunately, and as we show in this work this rarely is the case.

The rest of the paper is organized as follows. In section 2, we discuss RBS. In section 3, we present our3 methodology. In section 4, we evaluate RBS. In section 5, we review related work. Finally, we conclude and offer closing remarks in section 6.

2. ROW BUFFER SHARING (RBS)

In section 2.1 we discuss the organization of modern DDR2 SDRAMs. Note that while we focus on DDR2 SDRAM in this work, our technique can be used for other memory structures, e.g., DRAM or SDR SDRAM [9]. In section 2.2 we explain our motivation. In section 2.3 we discuss implementation details.

2.1 Memory background

A double data rate (DDR) SDRAM is composed of several independent memory banks. Figure 1 depicts DDR2 SDRAM details. Each bank is a two dimensional array of memory cells. To read from or write to memory, an entire row of a bank should be activated. An active command moves the row from a bank to the associated row buffer and opens the row. Once the row is opened, any number of read and write commands can be issued to transfer columns within the opened row. Row activation is a destructive operation, and the buffer should ultimately be written back to the memory array. A precharge command closes the row and restores it back to the memory making it ready for future operations.

Having several banks in memory increases the number of page hits. A page hit occurs when the requested row has already been opened making an active command unnecessary. Each row has several columns. The column width is equal to the number of data pins. Data is transferred on both rising and falling edges of the DDR2 SDRAM clock. A more detailed overview of DDR2 SDRAMs can be found in [3].

The memory controller is the interface between the on chip processor and caches, and the off-chip memory. The memory controller receives memory requests (read or write operations) and translates them into memory commands, e.g. row activation, column read/write, and row precharge. The Memory controller includes the memory scheduler which reorders and interleaves memory requests to reduce memory latency [4]. The memory controller must generate commands that conform to timing and resource constrains of the memory array. For example, in MT47H128M4B6-5E DDR2 SDRAM [3], the read or write column command should be issued at least tRCD (15ns) cycles after the active command. Table 2 illustrates timing characteristics of the MT47H128M4B6-5E memory used in this work.

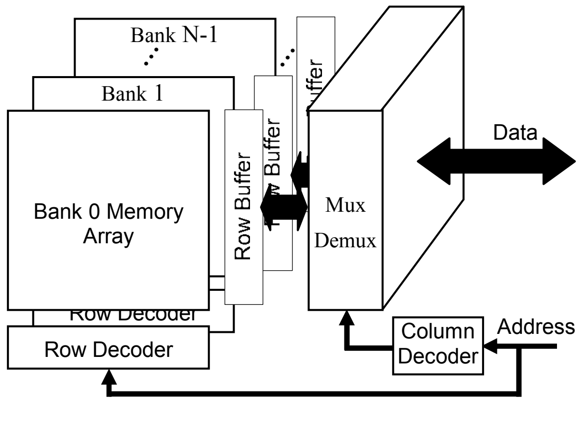

Figure 1. DDR2 SDRAM organization.

2.2 Motivation

A memory access that results in a row buffer hit only requires a column access. However, a row buffer miss needs precharge and active commands in addition to column access. As Table 2 shows, the latency of active and precharge commands (tRCD and tRP respectively) are 15ns each. In a 4-GHz processor, this is equal to 60 processor cycles. As a result, concurrent accesses to different rows of the same bank would be significantly slower compared to concurrent accesses to a single row of a bank or to rows of different banks. Therefore, average memory access time would depend on the state of the memory banks' buffers and the program's memory access pattern.

No matter how well a memory scheduler distributes memory requests over the banks, it can not prevent row conflicts (e.g., those resulting from a sequence of memory requests accessing different rows of the same bank). Such conflicts require switching rows and can potentially degrade performance if such conflicts occur too frequently.

In a memory system using 4 banks, e.g. MT47H128M4B6-5E, and assuming a uniform access distribution, each bank is accessed 25% of the time. In other words, 75% of the time, each bank and its associated row buffer are idle. In RBS, we borrow such idle row buffers for the active bank. As a result the borrowed row buffers act as a cache for the active bank. We assume that this process is managed by the memory controller. The Memory controller uses the extra row buffers to avoid row conflicts and to reduce memory latency.

We will benefit from RBS only if idle banks remain idle for a sufficient number of memory requests, and so possible benefits outweigh the overhead associated with precharging and activating.

Figure 2 shows memory request distribution for the crc32 benchmark from MiBench suites and for load/store instructions. The Y axis shows the bank number, and the X axis shows memory requests. For each memory request, we record the bank number corresponding to the requested memory address. In a DDR2 SDRAM with four banks, this number can be zero, one, two, or three. As the figure shows, during program runtime, the program goes through several phases. During each phase, the processor usually accesses one of the banks leaving the other three banks idle for most of the period. For example, from memory access 1536 to memory access 1804, bank number 1 is the only active one for more than 95% of the accesses. This period is equivalent to 6755 memory cycles or 67550 processor cycles (refer to table 1 for relative speed of CPU and memory).

Figure 3 shows the accumulative distribution of reusability distance for MiBench benchmarks studied in this work. Reusability distance shows the number of memory accesses occurring between two memory requests made to the same memory location. The Y-axis depicts accumulative value for reusability distance. The X-axis shows reusability distance in logarithm of 2 scale. To be precise, values from zero to ten in the X-axis correspond to reusability distances from zero to 1023.

As reported, distribution of reusability distance increases sharply when the distance changes from zero to three. It increases steadily for distances from three to 1023. 8% of memory accesses have a reusability distance of greater than 1023.

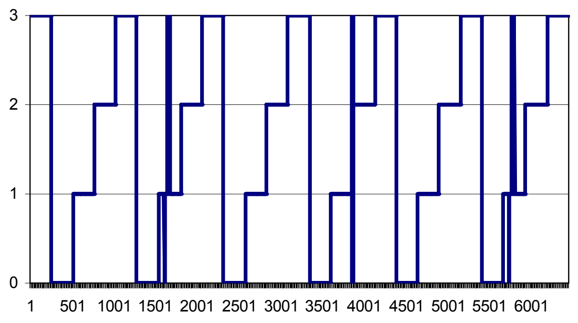

Figure 2. Cyclic access to banks in memory requests.

Only 30% of accesses have zero reusability distance. This means that in the baseline DDR2 SDRAM which uses one buffer per bank, only 30% of accesses result in a row hit. 32% of other accesses have reusability distance of one. In other words, if the number of bank row buffers is increased to two, the total hit rate will reach 62%. Following the same line of argument, a DDR2 SDRAM with four buffers per bank would have a 70% hit rate. In other words, four buffers cut the gap between a memory with one row buffer per bank and a memory with unlimited buffers per bank by half. We conclude from figure 3 that exploiting four buffers per bank would improve memory hit rate considerably.

Using four buffers per bank would require 16 buffers in the DDR2 SDRAM (which exploits four banks) studied in this work. This is not a reasonable choice for embedded processors as it would require a highly complex switching system. In addition, it would result in a higher number of idle row buffers during bank idle periods.

As presented in figure 2, quite often, only one of the four banks is active. This observation indicates the possibility of achieving high hit rates by sharing the four existing row buffers among the four banks. This is the result of the fact that often three of the four banks are idle and can let the active bank use their row buffers.

Note that further increases in the number of row buffers after sharing four rows would not result in any significant improvement. For example, doubling the number of buffers from four to eight increases the hit rate by less than 4%.

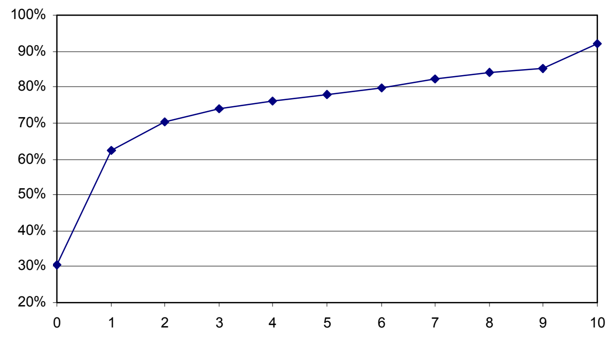

Figure 3. Accumulative distribution of reusability distance.

2.3 Implementation

In RBS, all row buffers are shared among the banks. Therefore, each bank can use up to four buffers simultaneously.

We use a crossbar interconnect to connect banks and row buffers. We assume one memory cycle for crossbar interconnect delay. The baseline DDR2 SDRAM uses a multiplexer to select one buffer to read or write. The crossbar switch uses four multiplexers to connect the four banks to the four row buffers. As such, we believe that the hardware overhead of the crossbar switch is reasonable in an embedded system.

The memory scheduler should select one row buffer if multiple row buffers are available. Also, the memory scheduler should evict one row, if all row buffers are in use. We use a combination of LRU (least recently used) [14] and unmodified policy [14] to select the row buffer for replacement. LRU policy selects the victim row buffer among unmodified ones. The reason that we exclude dirty row buffers from replacement is to minimize the latency. A dirty row buffer should be precharged, which incurs extra latency. If all the buffers are modified, the oldest modified buffer is selected for replacement.

3. METHODOLOGY

To simulate our technique, we use Simplescalar v3.0 [10]. We have modified the simulator to model a processor similar to Xscale [5]. Table 1 shows the XScale configuration. We used programs from MiBench [13] embedded benchmarks suits compiled by GNU's gcc v2.9 with –O3 optimization level (Table 2). We run each simulation for 500M instructions or up to completion, whichever comes first.

Table1. Base configuration for simulated processor

ROB/LSQ size	8/8
Branch Predictor	Bimodal predictor-128-entry
BTB	128-entry, 1-way
Scheduler	8-entry, RUU-like
Fetch Unit/ Fetch Buffer	1 instruction per cycle/2-entry
Decode, Issue, and commit width	2 instructions/ cycle
L1 – I/D Cache	32K, 32-way SA, 32-byte blocks, 1 cycle hit latency
Unified L2	N/A
ALU	1 int ALU, 1int mult/div 1 FP ALU, 1FP mult/div
Relative clock frequency of CPU to memory	10
TLB-I/D	32/32-entry, fully associative

We used a cycle accurate model for DDR2 SDRAM memory. We accurately model memory controller, bank conflicts, and bandwidth limitation. The model includes all details for timing parameters, delay between commands, and refreshing intervals. Table 2 presents the DDR2 SDRAM parameters used in this study. All of these parameters are extracted directly from the

MT47H128M4B6-5E DDR2 SDRAM datasheet [3]. The memory controller follows all timing constrains of table 2 when it issues a command to the banks.

Table 2. MT47H128M4B6-5E DDR2 SDRAM timing parameters [3].

t_{CK}	Clock Cycle	5 ns
t_{RCD}	Activate to read	15 ns
t_{CCD}	Read to read or write to write	10 ns
t_{WTR}	Write to read	10 ns
t_{RP}	Precharge to activate	15 ns
t_{RAS}	Activate to precharge	40 ns
t_{RFC}	Max refresh to refresh	70000 ns

4. RESULTS

Figure 4 reports hit rate improvement for RBS compared to the conventional DDR2 SDRAM. RBS improves memory access latency by assigning several buffers to an active bank. As such, several rows of a bank are opened at the same time, and this increases the likelihood that the row corresponding to a memory access has already been opened. The hit rate improvement changes from 18% in lame to 37% in crc32. On average, RBS increases row hit rate by 30%.

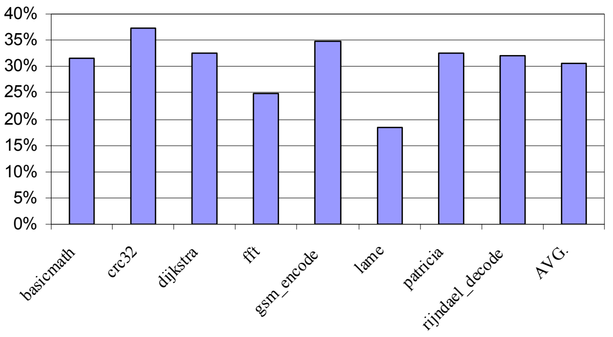

Figure 4. Row buffer hit rate improvement by RBS.

Figure 5 shows memory latency improvement for RBS compared to the baseline DDR2 SDRAM. Consistent with the data reported in figure 4, RBS improves memory latency significantly. When memory controller accesses an open row, it does not issue precharge and active commands, and this reduces the latency of memory. RBS reduces memory latency from 6.5% for lame to 16% for cr32. On average, memory latency is reduced by 12.7%.

In figure 6, we report processor overall performance improvement. As reported, we improve average processor performance by 7.6%. For some benchmarks, memory latency and processor performance react differently to RBS. For example, some benchmarks with lower memory latency reduction show better performance improvement. This could be explained by the variations in benchmarks performance sensitivity to memory latency. For some benchmarks, memory instructions have a higher impact on overall performance. Therefore, despite their lower memory latency reduction, the overall processor performance improvement is higher.

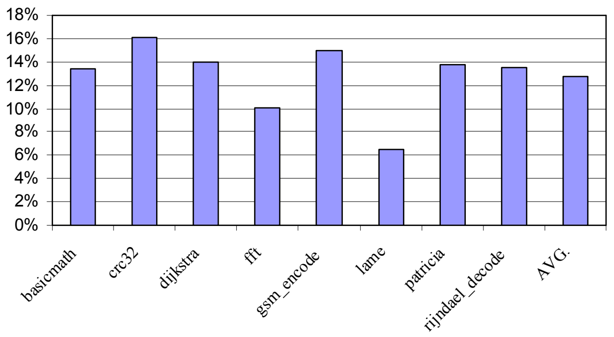

Figure 5. Average memory latency reduction by RBS.

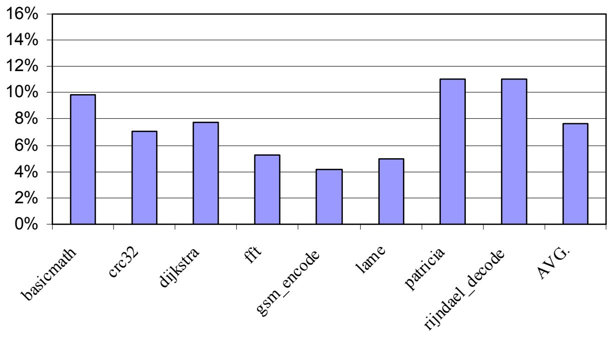

Figure 6. Processor performance improvement by RBS.

In all results reported so far, we have assumed a CPU with a clock frequency 10 times faster than the memory clock frequency (refer to Table 1). In figure 7 we report how RBS reacts to alternative relative CPU and memory speeds. We report RBS sensitivity to relative CPU and memory clock frequencies changing from four to 20. As the figure shows, performance improvements achieved by RBS increase as the gap between CPU and memory speed increases. This is due to the fact that the memory latency gap between RBS and the baseline scheme increases as CPU becomes increasingly faster than memory. On average, performance improves from 3.7% to 11.2% when relative speed varies between four and 20.

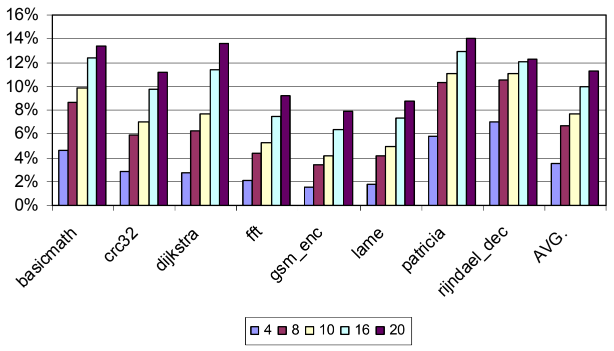

Figure 7. Processor performance improvement when relative speed of CPU and memory changes.

5. RELATED WORK

Rixner et al. [4] proposed a scheduling method that exploits locality of memory requests to reduce memory latency. They reordered memory references to increase the number of requests that access the same row of a bank. They do so by increasing the priority of requests that refer to the current open row. As such, they reduce row buffer replacement frequency and improve memory latency.

Moyer [11] developed and analyzed algorithms to perform access reordering statically at compile time. Moyer used loop unrolling and instructions reordering to improve memory locality. Moyer's technique applies specifically to stream-oriented workloads in cacheless systems. McKee et al. [12] used a runtime approach to order the accesses to streams in a stream memory controller. In this approach the memory controller considers each stream buffer in a round-robin fashion, streaming as much data as possible to the current buffer before going to the next buffer. This approach may reduce conflicts among streams, but it does not reorder references within a single stream.

Valero et al. [2] proposed a hardware-based mechanism to avoid bank conflicts in vector processors by accessing vector elements out of order. The Impulse memory system introduced by Carter et al. [6] improves memory system performance by dynamically remapping physical addresses. This approach requires modifications to the applications and operating system.

Alexander and Kedem [8] proposed a memory-based prefetching scheme to improve performance. They use a prediction table to store up to four possible predictions for any given memory address. All four predictions are prefetched into SRAM buffers. The size of their prediction table is kept small by using a large prefetch block size.

Our work is different from above as we increase the number of simultaneously opened rows to improve concurrency in main memory.

Researchers from NEC proposed exploiting virtual channels to reduce performance gap between processor and main memory [7]. Virtual channel memory (VCM) uses several channels between the memory cells and the memory I/O to store data temporarily. VCM increases the level of parallelism as memory data requests are prepared in separate channels while reading or writing the current data. Rixner [14] proposed different scheduling algorithm to exploit virtual channels for reducing memory latency. RBS is different from VCM as we exploit row buffers in phases that their corresponding banks are idle. In addition, we do not incur extra channels to cache bank rows which is critical to keep complexity and power affordable in embedded processors.

6. CONCLUSION

In this work we used periodic behavior in memory accesses to improve memory latency in embedded processors. We were motivated by the need for low memory latency and took advantage of application behavior and dynamic allocation opportunities available in the internal structure of modern DDR2 SDRAMs.

The cyclic and clustered nature of memory bank accesses provides an opportunity to borrow idle row buffers for active banks. We reduce resource conflicts for row buffers and decrease the number of precharge and active commands issued by the memory scheduler. Accordingly, we reduce average memory latency and improve performance by 12.7% and 7.6% respectively. In addition, we show that under possible future gap increasing between memory and CPU speed, RBS could achieve higher performance improvement.

7. REFERENCES

[1] W.A. Wulf and S.A. McKee, "Hitting the Wall: Implications of the Obvious", Computer Architecture News, vol. 23, no. 1, pp. 20-24, Mar. 1995.

[2] M. Valero, T. Lang, M. Peiron, and E. Ayguade, "Conflict-Free Access for Streams in Multi-Module Memories", Technical Report UPC-DAC-93-11, Universitat Politecnica de Catalunya, Barcelona, Spain, 1993.

[3] Micron. 1Gb DDR2 SDRAM memory: MT47H128M4B6-5E, June 2006.

[4] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling", In Proceedings of the International Symposium on Computer Architecture, 2000.

[5] Liao, S. Strazdus, M.Morrow, K.E. Velarde, and M.A. Yarch, "An Embedded 32-bit Microprocessor Core for Low-Power and High-Performance Applications", IEEE Journal of Solid-State Circuits, Vol. 36, No. 11 (November 2001), pp. 1599-1608.

[6] J. Carter, W. Hsieh, L. Stoller, M. Swanson, L. Zhang, E. Brunvand, A. Davis, C.-C. Kuo, R. Kuramkote, M. Parker, L. Schaelicke, and T. Tateyama, "Impulse: Building a smarter memory controller", HPCA, 1999.

[7] NEC. 64M-bit Virtual Channel SDRAMdata sheet, October 1998.

[8] T. Alexander and G. Kedem, "Distributed prefetch buffer/cache design for high performance memory systems", In Proc. of the Second High Performance Computer Architecture Symposium, pp. 254-263, Feb. 1996.

[9] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, "A Performance Comparison of Contemporary DRAM Architectures", ISCA, 1999.

[10] D. Burger, T. M. Austin, and S. Bennett, "Evaluating Future Microprocessors: The SimpleScalar Tool Set", Technical Report CS-TR-96-1308, University of Wisconsin-Madison, July 1996.

[11] S. A. Moyer, "Access Ordering and Effective Memory Bandwidth", Ph.D. Dissertation, Department of Computer Science, University of Virginia, Technical Report CS-93-18, April 1993.

[12] S. A. McKee et al., "Dynamic access ordering for streamed computations", IEEE Trans. Comput., 49(11):1255-1271, 2000.

[13] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, 2001, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite", Available at http://www.eecs.umich.edu/mibench/.

[14] Rixner, S. Memory Controller Optimizations for Web Servers. In Proc. of the 37th Intl. Symp. on Microarchitecture (Portland, Oregon, Dec. 04 - 08, 2004). Micro '04. 355-366.