DDR3 SDRAM with a Complete Predictor

Vladimir V. Stankovic[1], Nebojsa Z. Milenkovic[1]

Abstract – **In the arsenal of resources for computer memory system performance improvement, predictors have gained an increasing role in the past years. They enable hiding the latencies when accessing cache or main memory. In our previous work we have shown how average latency of DRAM memories can be decreased using various predictors. In three our papers we have considered three such predictors - a Dead-time predictor, a Zero-live-time predictor, and an Open-page predictor. All these predictors were used in simulators that simulate DDR SDRAM memory. In this paper we have integrated all the predictors and added them to a simulator that simulates a contemporary DDR3 SDRAM memory, written by ourselves. The results confirm the efficiency in using predictors.**

Keywords – **DRAM, memory, latency, DRAM controller, DRAM controller policy, predictor, bank, row.**

I. INTRODUCTION

A desire for better potential utilization of processors, which are becoming faster and faster, demands a memory system with similar performances. A critical ring in the hierarchically organized memory system is the main memory, implemented with chips of dynamic memory (DRAM – Dynamic Random Access Memory). In order to achieve as large bandwidth as possible, the chips of contemporary DRAM memories are organized with several independent memory banks, allow memory accesses pipelining, and buffer the data from the last activated row in each bank. Although increasing the memory bandwidth, these solutions make the contemporary DRAM memories performances dependable on memory access patterns. Contemporary DRAM memories are not really random access memories, characterized with identical access times to all locations in them. They are actually three-dimensional memories, with banks, rows, and columns as dimensions. DRAM data access with row opening demands the following time: Tacc = Trp+Tra+Tca, where Trp is row precharge time, Tra is row activate/access time, and Tca is column access time.

Using of read and write commands with autoprecharge eliminates the precharge time when the next access occurs, reducing the access time to Tra+Tca. Data accesses into already opened rows eliminate the precharge time and row access time, reducing the access time to Tca. The result is that consecutive accesses to different rows into single memory bank have larger latencies than consecutive accesses into the same row. Performances maximization of DRAM memories demands minimization of participation of precharges and rows openings.

This makes that we can influence DRAM memory latency by controlling the data placement into banks and rows. This is the basis of papers in which address remappings are considered, which transform memory addresses into banks, rows and columns that optimize DRAM performances for certain memory access patterns [3, 4].

DRAM memory latency can be decreased if the opened row is closed before occurring of the next data access directed to the same bank, but to different row. In that way the precharge time Trp is being hidden, so the latency is practically reduced to Tacc = Tra+Tca. The latency could be additionally reduced to Tacc = Tca, by hiding the row access time. This demands the next row that is going to be accessed, to be opened in advance. In-time closing of the opened row demands a prediction *when* to close the opened row. Opening in advance the next row to be accessed demands a prediction *which row* to open and *when*.

Papers [1, 2] deal with possibilities to predict the moment when the data block in the cache memory is to be declared 'dead' (i.e. unnecessary present in the cache, because it is not to be used in the near future) and when and which data block to fetch to the cache in advance. This inspired us to investigate the possibilities of applying some of those ideas on DRAM memory performance optimization. In this paper we have restrained on ideas from [1], which relate to applying metrics of characteristic time parameters of data blocks transferred to cache memory. Analogically, we have defined proper characteristic time parameters for DRAM memories. By simulation, we have concluded that DRAM memory accesses have some regularity that can be used for prediction when to close the opened row, and which is the next row to be opened. Based on those results, we have proposed three predictors. Two of them predict when to close the opened row, and the third one predicts the next row to be opened. In our previous papers [5, 6, 7] we have simulated using these predictors with a DDR SDRAM memory. In this paper we try using them on a contemporary DDR3 SDRAM memory. The simulator for this memory was written by ourselves.

DDR3 SDRAM memories are the most advanced type of commodity SDRAM memories. They have several new features which enable improving the control of them and increasing their performances, mainly through higher bandwidth. DDR3 SDRAM devices support posted CAS commands, which allows a DDR3 SDRAM memory controller to treat a row activation command and a column access command as a unitary command pair to be issued in consecutive cycles. This is simpler then issuing two separate commands which must be properly controlled and timed, as in DDR SDRAM. In addition, DDR3 SDRAM devices of all capacities have at least 8 banks of independent DRAM arrays that increase the capacity of sens amplifiers as buffers with reduced access time. Also, DDR3 devices can work at clock frequency range from 300÷800 MHz, with transfer rates up to 1600 MT/s. These changes influence more positively on bandwidth increase than on latency reduction defined by

[1]Authors are with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Nis, Serbia. E-mail: [vladimir.stankovic, nebojsa.milenkovic]@elfak.ni.ac.yu

characteristic time parameters Trp, Tra, Tca, what causes increase of latency relative participation in the complete time needed for a data block transfer. For example, a DDR3 SDRAM Micron MT41J128M8 memory [9] has the following parameters: fcm = 800MHz, Tcm = 1.25ns, data transfer rate: 1600MT/s, Trp = Tra = Tca = 12.5ns. For a Wdb=64B data block read to be performed, the DRAM module with Wm=64lines=8B, a single burst with a length of Lburst=Wdb/Wm=8 is needed. The time needed for sending such data block to the controller, in the best case is:

Tca + Lburst x 1/2 x Tcm = 10Tcm + 4Tcm = 14Tcm, with a latency participation of 10/14 = 0.714, or 71.4%.

In the worst case this time is Trp + Tra + Tca + Lburst x 1/2 x Tcm = 30Tcm+4Tcm=34Tcm, with a latency participation of 30/34=0.882, or 88.2%.

The paper is organized as follows. In section II the basic idea, and in section III the predictors' design and implementation, are exposed. Section IV contains the used simulation model, and section V gives a review of the obtained results. Section VI is the conclusion.

II. BASIC IDEA

A classic DRAM controller uses two possible policies: Open Row and Close Row. When using the first one, the row is kept opened, which gives latency of Tca if the next DRAM access is directed to the opened row, and Trp+Tra+Tca, if the next DRAM access is directed to some other row. When using the second policy, a row is being closed after every access, so the latency is always the sum Tra+Tca. The first policy gives good results with programs that have good memory access locality, and the second one with programs whose DRAM accesses have random character. Our goal is to achieve a policy more efficient than the both policies for both types of programs. This can be achieved if the opened row is kept open for as long as there are accesses into it, and then closed after the last access into it. After that the next row to be opened should be predicted and opened in advance.

Let us define the metrics from [1] related to DRAM. *Live time* is a time interval that elapses from opening the row in a bank until the last access into that row before its closing. *Dead time* is a time which elapses from the last access to an open row until the moment of its closing. *Access interval* is a time interval which elapses between two consecutive accesses to an open row in a bank. A live time of an open row is called a *zero live time*, if after its opening there are no further accesses to that row till its closing.

In this paper we consider a DRAM controller with 2 predictors: a Close-page predictor, and an Open-page predictor. First the Close-page predictor predicts when to close the currently opened DRAM row. After that, the Open-page predictor predicts the next row to be opened. In case of accurate predictions the latency time is reduced to only Tca.

The mentioned Close-page predictor consists of two predictors: a Zero-live-time predictor and a Dead-time predictor. The first predictor is used always when a new row is opened, and it predicts whether its live time will be a zero live time or not. If yes, that row is closed immediately after completing the DRAM access. If not, the row is kept opened and after that access, and during further accesses the Dead-time predictor is used to predict whether that row has entered its dead time. If it has, the row is closed, if not it is kept open.

In case of a prediction that closes the row (either by the Zero-live-time or by the Dead-time predictor) the Open-page predictor is activated. This predictor consists of two tables - Row History Table and Pattern History Table. Based on them, the next row to be opened is predicted, and then opened.

In next section all the predictors are described in detail.

III. PREDICTORS' DESIGN AND IMPLEMENTATION

The Zero-live-time predictor uses two bits for each row in the DRAM. Those two bits are used as a saturated counter, with values from 0 to 3. Every time a zero live time occurs the counter is incremented, except its previous value was 3. Every time a nonzero live time occurs the counter is decremented, except its previous value was 0. When predicting, it is predicted that the live time will be a zero live time if the counter's value is 2 or 3, i.e. nonzero live time if the counter's value is 0 or 1. The starting counter's value is 0.

Implementation of this predictor is simple. It may be in a form of a SRAM memory with suitable organization integrated into the DRAM controller, since number of rows in the system may have large values. For example, a DDR3 DRAM chip that has 8 banks with 8K rows each, demands 128Kb or 16KB. Changing the values of the counter (incrementing, decrementing, resetting) is done by read-modify-writes. When predicting, a read is performed, and depending on the value being read, the controller will issue commands with autoprecharge, or not.

The Dead-time predictor is based on access interval time values. Our simulation results showed that the average dead time is several times larger than the average access interval time, so that fact is used for dead time prediction. When a value that is the last access interval time, multiplied by 2, elapses, it is predicted that the row has entered its dead time. So the only value that is being taken care of is the last access interval time. We used one register for storing the access interval time for each bank in the system.

The implementation of the Dead-time predictor demands the DRAM controller to have one counter for each bank (to take care of the elapsed time since the last access), one register for each bank, for storing the last access interval value, and one comparator for each bank (for comparing the access interval register value with the counter). In order to minimize the counters' length, they could be triggered with a signal derived by dividing the DRAM's clock. A simple shift operation by 1 position over the access interval register would be needed for defining the boundary value. By comparing this value with the counter the controller would decide whether to issue a precharge command or not. A controller that implements Open Row Policy already has a register for each bank for storing the last open row index, and a comparator for comparing the current access row index with that register. So the hardware added to a DRAM controller which implements the Dead-time predictor would have similar complexity as the mentioned implementation of Open Row Policy.

The structure of the Open-page predictor is presented in Figure 1. It consists of two tables – Row History Table (RHT), and Pattern History Table (PHT). RHT stores the last k rows that were activated in each of the banks, and PHT contains the predictions. PHT has m ≤ n items, where n is number of bank rows. Each item contains j two-part fields: row and next predicted row (r_k and r_{nxt}). PHT access index is obtained as t least significant bits of the sum (trancated addition) of the last k row indeces from the proper item for that bank in RHT, so $m=2^t$.

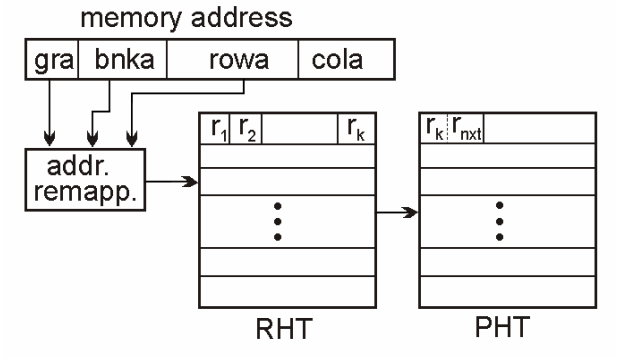

Fig. 1. Structure of the Open-page predictor

Implementation of the Open-page predictor would demand $g \cdot b \cdot k \cdot \lceil \log_2 n \rceil$ bits for RHT (g is the number of DRAM chip ranks, b is the number of banks per rank) and $m \cdot j \cdot 2 \cdot \lceil \log_2 n \rceil$ bits for PHT. Also, one t-bit adder and a multiplexer of type (k,1)×t are needed, for a control block implemented as a finite state machine. For the adopted DRAM structure of 2GB with k=4, m=4096 and j=2, 832B are needed for RHT and 26KB for PHT.

IV. System simulation model

For simulation we have used the program Sim-Outorder from the Simplescalar Tool Set [8]. We have integrated this simulator with a program that simulates DDR3 SDRAM memory, written by ourselves. This integrated simulator performs an execution-driven simulation, which is much more accurate than trace-driven simulations. The characteristics of the simulated processor are: a superscalar processor that issues at most 4 instructions on every clock cycle and supports out of order instruction execution, uses a two-level branch predictor, and has two levels of cache memories. The first one contains separate instruction and data caches. They are both 16KB large, use direct mapping and have line size of 32B. The second level contains a unified cache, 2MB large, uses set-associative mapping with 4 lines per set, and have line size of 128B. All the cache memories use write-back policy. The processor clock frequency is 3.2 GHz.

The simulated memory has the most recent characteristics: two 128 data line ranks of 1Gb DDR3 SDRAM devices, 8 banks/chip, 8K rows/bank, row capacity is 1K×2B, Trp, Tra, Tca are 40 processor clock cycles each.

We have simulated executions of 6 benchmark programs from the SPEC95 suite: cc1, compress, ijpeg, li, m88ksim, and perl [4].

V. Results

As a start, we have measured the same parameters as when using DDR SDRAM in our previous papers [5, 6, 7]: open row hit probability, number of accesses with zero live times, number of accesses with nonzero live times, and average values for access interval time, live time and dead time, measured in processor clock cycles. We wanted to see if DDR3 SDRAM would also show performance improvements when using predictors, and this can be predicted from the above parameters. The results are shown in Table 1. It can be seen that in benchmark programs with small open row hit probabilities (cc1, ijpeg, perl) the number of zero live times is much greater than the number of nonzero live times, which is reasonable. In benchmarks with large open row hit probabilities (compress, m88ksim, li) there are much more nonzero live times than zero live times. It can also be noticed that in all 6 cases, the average value of access interval time is several times smaller than the average value of dead time. These results are pretty similar to those for DDR SDRAM, which suggests that both Zero-live-time predictor and Dead-time predictor would yield latency decrease, and an efficient Close-page predictor opens a chance for having a good Open-page predictor, too.

TABLE I
MEASURED PARAMETERS OF BENCHMARK PROGRAMS

Benchmark	compr.	m88.	li
Open row hit probability	0.91	0.89	0.80
Zero live times	53	126	64
Non zero live times	342	497	180
Access interval	1451	150798	61623
Live time	8816	1469739	343359
Dead time	33233	8355280	18244638

Benchmark	cc1	ijpeg	perl
Open row hit probability	0.37	0.33	0.07
Zero live times	57649	28942	1184977
Non zero live times	14318	1822	38743
Access interval	25403	13796	19740
Live time	75564	113631	49376
Dead time	309924	144892	28562

Table 2 shows the prediction accuracies of the three predictors: Zero-live-time (ZLT), Dead-time (DT) and Open-page (OP). The accuracies of the Dead-time predictor are solid, and as expected. The accuracies of the Open-page predictor are pretty good, with the exception of li. The accuracies of the Zero-live-time predictor for the three benchmark programs with small open row hit probabilities (cc1, ijpeg, perl) are excellent, and the accuracies of the same predictor for the three benchmark programs with large open row hit probabilities (compress, m88ksim, li) are pretty bad. However, these bad prediction accuracies can be deceptive. First of all, in these 3 benchmark programs there are very little situations with zero live times (53, 126, and 64) so high

accuracies are difficult to obtain. Second, when having such small numbers, one should look not just for the prediction accuracies themselves, but for the number of total hits and misses, too. These numbers of total hits/misses for the Zero-live-time predictor are given in Table 3. It can be seen in this table that the total numbers of the zero live time mispredictions are negligible (3, 9, and 2). From that we can conclude that the Zero-live-time predictor, for these three benchmark programs, although having rather low prediction accurices, will not have a significant negative influence on DRAM latency.

TABLE II
PREDICTION ACCURACIES OF THE PREDICTORS

Benchmark	comp.	m88.	li	cc1	ijpeg	perl
ZLT pred.	0	0.25	0.33	0.83	0.98	0.97
DT pred.	0.50	0.68	0.58	0.65	0.55	0.81
OP pred.	0.83	0.59	0.21	0.65	0.82	0.88

TABLE III
PREDICTION HITS/MISSES OF THE ZERO-LIVE TIME PREDICTOR

Benchmark	comp.	m88.	li	cc1	ijpeg	perl
ZLT hits	0	3	1	49164	24554	1151
ZLT misses	3	9	2	10252	408	33

(Numbers for perl in Table III are given in thousands.)

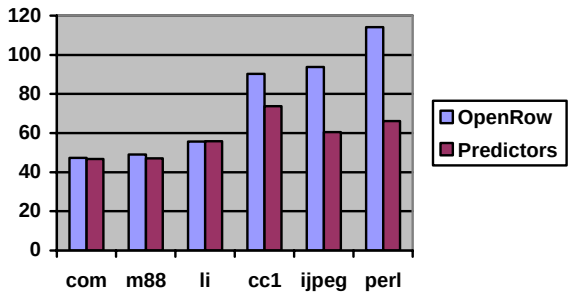

Fig. 2. Average DRAM latencies in processor clock cycles

Figure 2 shows the average DRAM latencies, in processor clock cycles, for the Open Row Policy and the hybrid policy which uses the three predictors. It can be seen that in all the cases the predictors either gain improvements or does not spoil the latency. In the three benchmark programs with good open row hit probabilities (compress, m88ksim, li) the improvements are negligible. The fact that there are no exacerbations in these cases can be considered a success. It is very difficult to obtain improvements in these programs, since improvements are possible only when the opened row is changed, and this happens very rarely in these programs. This can be corroborated with the fact that the theoretical latency minimum that can be obtained is Tca, which is 40 cycles, and it can be seen that Open Row Policy itself gives latencies of about 45-55 cycles. In programs with small open row hit probabilities (cc1, ijpeg, perl) there are visible improvements when using the predictors. The latency decrements for these benchmark programs are 18% for cc1, 36% for ijpeg and even 42% for perl. Another thing should be pointed out. The Close Row Policy gives rather predictable DRAM latencies of Tra+Tca. So for all the benchmark programs, the latencies when using the Close Row Policy should be around 80 cycles. If we compare this with the obtained DRAM latencies when using the three predictors, it can be seen that for all 6 programs the predictors give better results than both the Open Row and the Close Row Policy, which was our main goal.

VI. CONCLUSION

In this paper we have considered performances of contemporary DDR3 SDRAM that uses two predictors which predict when to close the opened DRAM row, and one predictor which predicts the next row that should be opened. The considered solution gives performance improvements, both compared to Open Row and Close Row policy for all the used benchmark programs. An implementation of such a solution could demand proper changes in the DRAM controller. The price increase of such a controller accompanies the tendencies toward more complex DRAM controllers, and could be easily accepted in the near future.

REFERENCES

[1] Z. Hu, S. Kaxiras, M. Martonosi, “Timekeeping in the Memory System: Predicting and Optimizing Memory Behavior”, The 2003 IEEE International Solid_state Circuits Conference (ISSCC 2003), February 2003.

[2] A. Lai, C. Fide, B. Falsafi, “Dead-Block Prediction and Dead-Block Correlating Prefetchers”, Proc. 28th ISCA, June 2001, pp. 144-154.

[3] Z. Zhang, Z. Zhu, X. Zhang “A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality”, Proc. 33rd AIS on Microarchitecture, (Micro-33), Monterey, Calif. 2000.

[4] V. Stankovic, N. Milenkovic, “Access Latency Reduction in Contemporary DRAM Memories”, Facta Universitatis, series: Electronics and Energetics, Vol. 17, No. 1, April 2004, pp. 81-97.

[5] V. Stankovic, N. Milenkovic, “DRAM Controller with a Simple Predictor”, ICEST 2005, Nis, June 29 - July 1, 2005, Vol. 2, pp. 449-452.

[6] V. Stankovic, N. Milenkovic, “DRAM Controller with a Close-Page Predictor”, Eurocon 2005, Belgrade, November 2005, pp. 693-696.

[7] V. Stankovic, N. Milenkovic, “DRAM Controller with a Complete Predictor”, ETRAN 2006, June 2006, Vol. III, pp. 34-37.

[8] D. Burger, T. M. Austin, “The SimpleScalar Tool Set, Version 2.0”, University of Wisconsin-Madison Computer Sciences Department Technical Report #1342, June 1997.

[9] Micron 1Gb DDR3 SDRAM MT41J128M8 Data Sheet, http://www.micron.com/products/dram/ddr3/