DRAM Controller with a Simple Predictor

Vladimir V. Stankovic[1], Nebojsa Z. Milenkovic[2]

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 paper [1] it is shown how temporal parameters of cache memory access, defined as live time, dead time and access interval could be used for prediction of data prefetching. In this paper a possibility of applying an analog technique on controlling DRAM memory opened rows closing, is being researched. Obtained results confirm such a possibility, which served us to propose a simple predictor. Using such a predictor can significantly decrease the average DRAM latency.**

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:

$$Ta = Tpr+Tra+Tca \tag{1}$$

where

- Tpr – is precharge time,
- Tra – is row access time,
- Tca – is column access time.

Using of read and write commands with autoprecharge eliminates the precharge time when 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. Performance maximization of DRAM memories demands minimization of participation of precharges and rows openings.

This makes that we can influence DRAM memory performances (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 just before the next data access directed to the same bank, but to different row, occurs. In that way the precharge time Tpr is being hidden, so the latency is practically reduced to Ta = Tra+Tca. The latency could be additionally reduced to Ta = 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 prefetch to the cache. Those ideas maybe could be applied to DRAM memories. That 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 regularities that can be used for prediction when to close the opened row. Based on those results, we have proposed a predictor, which could be integrated into existent DRAM memory controllers.

The paper is organized as follows. In section II we consider the existent DRAM controller policies and the basic idea. In section III the used simulation model is exposed. Section IV gives a review of the obtained results, and section V is the conclusion.

II. What prediction is based on?

A classic DRAM controller uses two possible policies (strategies): Open Page (Row) Policy (Optimistic Policy) and Close Page (Row) Autoprecharge Policy (Pessimistic Policy). When using the first one, the row is kept opened, which decreases the latency if the next DRAM access is directed to

[1]Vladimir V. Stankovic is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Nis, Serbia and Montenegro, E-mail: svlada@elfak.ni.ac.yu

[2]Nebojsa Z. Milenkovic is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Nis, Serbia and Montenegro, E-mail: nmilenko@elfak.ni.ac.yu

the same row as the previous, and increases the latency if the next DRAM access is directed to some other row. In the first case the latency is equal to Tca, and in the second it is equal to the sum Tpr+Tra+Tca. When using the second policy, a row is being closed after every access, so the latency is always the same – the sum Tra+Tca. The Open Row Policy gives good results if there is a good memory access locality, and the Close Row Autoprecharge Policy gives good results if DRAM accesses have mostly random character. In our previous papers [4, 5] we have already considered various possibilities of obtaining hybrid policies, which use the advantages of both policies. The goal is to achieve a policy more efficient than both the Open Row and Close Row Autoprecharge Policy, and in that way, decrease the DRAM latency. In ideal case the opened row should be kept open for as long as there are accesses into it, and not to some other row, and it should be closed after the last access into it. In that way the system would be prepared for the next row access. In that case the precharge time could be hidden every time the row is changed, which would decrease the latency. In this paper we consider a hybrid policy which strives to predict the moment when to close the opened row.

Since we want to apply the metrics analogous to those from [1] in order to improve DRAM memory performances, let us first define those metrics related to DRAM memory. 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 open row until the moment of its closing. Access interval is a time interval which elapses between two consecutive accesses to 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. If there is at least one access to already open row before its closing, then that row's live time is not a zero live time.

An insight of open row entering into dead time is a signal for the DRAM controller to close that row, and eventually open some other row in the same bank. It would also be preferable for the DRAM controller to be able to recognize (i.e. to predict) opening row with zero live time, since in that case that row should be immediately closed.

III. System simulation model

For simulation we have used the program Sim-Outorder from the Simplescalar Tool Set [6]. We have integrated this simulator with programs that simulate DRAM memories, written by ourselves. 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. The processor clock frequency is 2 GHz. As a branch predictor a two-level branch predictor was used. There are 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, 1MB large, with set-associative mapping - associativity of 4, and line size of 128B. All the cache memories use write-back policy.

The simulated DRAM memory has the following characteristics: there are 4 banks in one chip, 4096 rows in a bank, the row capacity is 1KB, the precharge time, row access time, and column access time are 20 processor clock cycles each, the memory bus has 128 data lines.

We have simulated executions of 6 benchmark programs from the SPEC95 suite: cc1, compress, ijpeg, li, m88ksim, and perl. The characteristics of those programs can be found in our previous papers [4, 5].

IV. Results

As a start, we have measured the following parameters: open row hit probability, number of zero live times, number of non zero live times, and average values for access interval, live time and dead time, measured in processor clock cycles. The results are shown in Table I. It can be seen that in benchmark programs with small open row hit probability (cc1, ijpeg, perl) number of zero live times is much greater than number of non zero live times, which is reasonable. In benchmarks with large open row hit probability (compress, li, m88ksim) there are much more non zero live times than zero live times. These results recommend to try researching the possibilities of designing a predictor which, when opening a row, predicts its live time as zero or nonzero. This will, probably, be a subject of our future work. If the other parameters are observed, it can be noticed that in all the cases, not dependable on the open row hit probability, the average value of access intervals is much less than the average value of dead times. This suggests a possibility of defining a simple predictor. If, from the last access to open row, a certain amount of time (equal to some boundary value) has elapsed, then that row should be closed, since it has probably entered the dead time. If that amount of time has not yet elapsed, the row is to be kept open. As a boundary, a value that is the same order of magnitude as the last access interval should be used. For instance, it could be the last access interval multiplied by 2 or 4.

Table I
Measured characteristics of benchmark programs

Benchmark	cc1	compress	ijpeg
Open row hit probability	0.34	0.84	0.31
Zero live times	58662	51	28895
Non zero live times	15811	587	2621
Access interval	65833	2889	28692
Live time	165718	17773	155700
Dead time	1243661	161594	580286

Benchmark	li	m88ksim	perl
Open row hit probability	0.76	0.83	0.07
Zero live times	59	101	1174201
Non zero live times	236	794	44955
Access interval	839903	643307	43078
Live time	3420116	3712389	97176
Dead time	18495793	19202689	135064

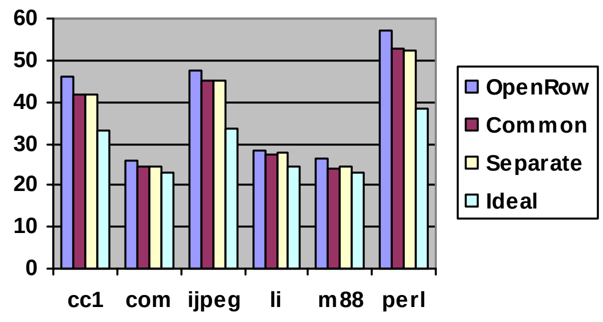

Fig. 1. Average latencies in processor clock cycles for rgbc

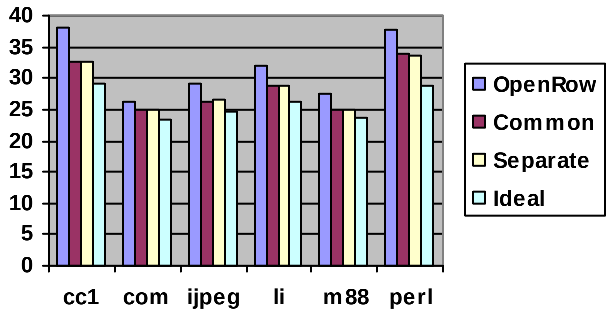

Fig. 2. Average latencies in processor clock cycles for rgrbcx

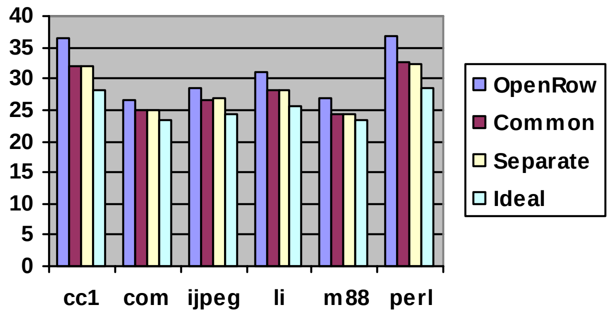

Fig. 3. Average latencies in processor clock cycles for rbrgcx

We have tried 2 variants for boundary levels – the last access interval multiplied by 2 and 4. The results are practically the same; i.e. the differences are insignificant. In this paper we show the results when the boundary value is equal to the access interval multiplied by 2. In paper [4] we have considered various address remappings, which increase the open row hit probability. In order to gain as objective evaluation as possible, we have tried 3 variants. The first one is a classical page interleaving scheme, which also can be named as row-group-bank-column (rgbc) by the sequence of meaning of the address bits, and the other two are the two address remappings from the mentioned paper that have showed as the best: rgrbcx and rbrgcx. For all of the named combinations we have tried two possible solutions. The first one uses only one common value of access interval, which is defined by every appearance of new access interval in any bank. In the second solution there is one value of access interval for each bank in the system.

The average DRAM latencies, in processor clock cycles, are shown in Figs. 1, 2 and 3. These Figs. show average DRAM latencies when using Open Row Policy (Open Row), policies with the proposed simple predictor with a common value and with separate values of access interval (Common and Separate), and a policy with an ideal predictor, i.e. a predictor whose prediction accuracy would be 100% (Ideal). It can be seen that the proposed solutions, although simple, give rather good improvements. The encouraging thing is that improvements are obtained not only for the basic rgbc scheme, but also for the both address remapping schemes. This means that it is possible to apply the address remappings, which already give respectable improvements, together with the considered predictor to additionally increase the improvements.

If we compare the solutions with a common value and with separate values of access interval, there are almost no differences among them. In the solution with a common value there are access interval interferences from different banks. That interference is removed when using separate values for each bank. This interference is not significant in a single program environment, which was the case of our simulations (this is why there are practically no differences in the results). In some cases (li and m88ksim with rgbc and ijpeg with rgrbcx and rbrgcx) worse results for Separate than Common could be explained by longer negative influences of extreme, relative to average, values of access interval. In a multiprogram environment access intervals of different programs can differ a lot. In that case the interference influence in the solution with a common value would dominate, which would certainly decrease the prediction accuracy, so this solution would show less performance improvement. We can conclude this from Table I, which shows that average access interval values for different programs can range up to 1 to 290 (for compress and li).

Table II shows the prediction accuracy and coverage when using one common register for all the banks. The coverage is the part of accesses for which the predictor made a certain prediction, starting from the first appearance of access interval value. The prediction accuracies and coverages when using separate values for each bank are very similar to these ones, so we omit them. In Table II cr (close row) is the probability of the accurate prediction that the row should be closed, and ncr (not close row) is the probability of the accurate prediction that the row should be kept open. The proper coverages are given in the parenthesis. By simple addition of these coverage percentages it can be concluded that the percentage of the accesses not involved by the predictor is negligible – in almost all the cases it is about 1% or less. Only in case of li using basic page interleaving scheme this percentage is about 5%. These accesses not involved by the predictor comprises all the first accesses which are zero live times, till the appearing of the first non zero live time, i.e. the first value for access interval, which is the moment when the predictor starts with the prediction process. If we see the prediction accuracies themselves it can be seen that in 29 of 36 cases they are more than 70%, and in 19 of 36 cases are more than

80%. These are rather good values. The high prediction accuracies also have high coverages in most of the cases. It happens, however, the predictions that the row should be kept open, to be very low, and to have rather high coverages, when using the basic page interleaving scheme, in the benchmarks with low open row probabilities (cc1 - 0.43 (63%), ijpeg - 0.34 (80%) and perl - 0.08 (78%)). These cases deserve an attention to look for improvements – probably the mentioned predictor which predicts zero live times could give needed improvements.

TABLE II
PREDICTION ACCURACY AND COVERAGE FOR COMMON

Benchmark	cc1	compress	ijpeg
cr – rgbc.	0.79 (37%)	0.63 (21%)	0.80 (20%)
ncr – rgbc	0.43 (63%)	0.99 (78%)	0.34 (80%)
cr – rgrbcx.	0.78 (48%)	0.66 (21%)	0.79 (25%)
ncr – rgrbcx	0.85 (52%)	0.97 (79%)	0.96 (75%)
cr – rbrgcx.	0.74 (48%)	0.68 (21%)	0.70 (24%)
ncr – rbrgcx	0.89 (52%)	0.98 (78%)	0.94 (76%)

Benchmark	li	m88ksim	perl
cr – rgbc.	0.60 (27%)	0.78 (18%)	0.95 (22%)
ncr – rgbc	0.96 (68%)	0.99 (80%)	0.08 (78%)
cr – rgrbcx.	0.78 (29%)	0.86 (18%)	0.82 (30%)
ncr – rgrbcx	0.90 (70%)	0.96 (81%)	0.72 (70%)
cr – rbrgcx.	0.73 (31%)	0.82 (19%)	0.82 (32%)
ncr – rbrgcx	0.94 (67%)	0.98 (80%)	0.77 (68%)

Let us now consider how the implementation of the considered predictor would influence the complexity and price of the DRAM controller. The controller should have a counter for each bank (to take care of the elapsed time since the last access), one common register for all the banks in the system or separate registers, one for each bank, (for storing the last access interval value), and one comparator for each bank (for comparing the access interval register value(s) with the counters). In order to minimize the length of the counters, they could be triggered with a signal derived by dividing the DRAM's clock. A simple shift operation by 1 or 2 positions over the access interval register(s) would be needed for defining the boundary value(s). By comparing this value with the counters the controller would decide whether to issue a precharge command or not. The controller that implements the Open Row Policy has a register for each bank for storing the last open row index, and comparators for comparing the index of the row to which the current access occurs with those registers. Compared to that, we could say that the controller with the proposed predictor would have similar complexity and price.

V. CONCLUSION

In this paper we have considered performances of DRAM memory with a controller that uses a simple predictor which predicts whether the opened DRAM row should be further kept open or it should be closed. The considered two solutions (the first one, with a common register, and the second one, with separated registers, for access intervals storing) are rather simple, and give good performance improvements, for all the three considered variants (the basic page interleaving scheme and the two address remapping schemes). The implementation of the considered solutions would be simple and with acceptable price. It is expected that the considered predictor, amplified with a predictor that predicts whether the live time is going to be a zero live time, would give additional performance improvements. This will be a subject of our further research. Also, we will focus our attention on designing a predictor which predicts which is the next row to be opened, after closing the current row. That kind of solution would additionally decrease the DRAM memory latency.

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] Zhang Z., Z. Zhu and 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, "Two New DRAM Controller Policies", Yu Info 2004, Kopaonik, March 2004.

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