LETTER

DDR3 SDRAM with a Complete Predictor

Vladimir V. STANKOVIC^{†a)}, *Student Member* and Nebojsa Z. MILENKOVIC[†], *Nonmember*

SUMMARY In the arsenal of resources for improving computer memory system performance, predictors have gained an increasing role in the past few years. They enable hiding the latencies when accessing cache or main memory. In our previous work we proposed a DDR SDRAM controller with predictors that not only close the opened DRAM row but also predict the next row to be opened. In this paper we explore the possibilities of trying the same techniques on the latest type of DRAM memory, DDR3 SDRAM, with further improvements of the predictors.

key words: DDR3 SDRAM, latency, predictor

1. Introduction

A desire for better 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 main memory, implemented with chips of dynamic memory (DRAM - Dynamic Random Access Memory). In order to achieve as large bandwidth as possible, chips of contemporary DRAM memories are organized with several independent memory banks, which allow memory-access pipelining and buffer the data from the last activated row in each bank. Although increasing the memory bandwidth, these solutions make DRAM performances dependant on memory access patterns. DRAM data access with row opening demands the following time: $T_a = T_{rp} + T_{ra} + T_{ca}$, where T_{rp} is row precharge time, T_{ra} is row activation time and T_{ca} is column access time. This time can be decreased if the opened row is closed before the occurrence of the next data access, directed to the same bank but to a different row. In this way, the precharge time T_{rp} is being hidden, so the latency is practically reduced to $T_{ra} + T_{ca}$. This latency can be additionally reduced to T_{ca} , by hiding the row activation time. This demands that the next row accessed be opened in advance.

In our previous paper [1], we proposed a DDR SDRAM controller with similar predictors that both close the opened DRAM row and predict the next row to be opened. In this paper we explore the possibilities of trying the same techniques on DDR3 SDRAM memories. DDR3 SDRAM memories [2], [3] are the most advanced type of commodity SDRAM memories. They have several new features which enable improving their control and increasing their performances, mainly through higher bandwidth. DDR3 SDRAM

devices support posted CAS commands, which allows the 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 than issuing two separate commands which must be properly controlled and timed, as in DDR SDRAM. In addition, DDR3 SDRAM devices have at least 8 banks of independent DRAM arrays that increase the capacity of sense amplifiers with reduced access time. Also, they can work at clock frequencies up to 800 MHz, with transfer rates up to 1600 MT/s.

These changes have a more positive influence on bandwidth than on reducing the latency of characteristic time parameters T_{rp} , T_{ra} , T_{ca} , which causes increase of the latency relative participation in the complete time needed for a data block transfer. For example, a DDR3 SDRAM memory from [3] has the following parameters: $f_{cm} = 800$ MHz, $T_{cm} = 1.25$ ns, bandwidth: 1600 MT/s, $T_{rp} = T_{ra} = T_{ca} = 12.5$ ns. For a $W_{db} = 128$ B data block read to be performed from the DRAM with $W_m = 128$ lines = 16 B, a single burst with a length of $L_{burst} = W_{db}/W_m = 8$ is needed. The time needed for sending such data block to the controller, in the best case is:

$T_{ca} + L_{burst} \times 1/2 \times T_{cm} = 10 T_{cm} + 4 T_{cm} = 14 T_{cm}$, with a latency participation of $10/14 = 0.714$, or 71.4%.

In the worst case this time is $T_{rp} + T_{ra} + T_{ca} + L_{burst} \times 1/2 \times T_{cm} = 30 T_{cm} + 4 T_{cm} = 34 T_{cm}$, with a latency participation of $30/34 = 0.882$, or 88.2%.

This shows that DRAM latency presents a significant factor in the overall delay of contemporary DRAMs.

The main advances of this paper compared to [1], besides trying our predictors on DDR3 SDRAM, are:

1. We try a modification of the zero live time predictor and gain the same performance improvements with 16 times less hardware.
2. We try a smaller version of the open page predictor, which requires 4 times less hardware and yields the same performance improvements.
3. We use a larger set of benchmark programs.

The paper is organized as follows. In Sect. 2 we describe the used simulation model. Sections 3, 4 and 5 contain the results for the dead time predictor, the zero live time predictor and the open page predictor, respectively. Section 6 is the conclusion.

2. System Simulation Model

For simulation we have used Sim-Outorder from Sim-

Manuscript received May 4, 2010.

[†]The authors are with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Nis, Serbia.

a) E-mail: vladimir.stankovic@elfak.ni.ac.rs

DOI: 10.1587/transinf.E93.D.2635

plescalar [4]. We have integrated this simulator with a DDR3 SDRAM simulator, written by ourselves. Characteristics of the simulated system are basically the same as in [1]. The main difference is the increased processor clock frequency, which is 3.2 GHz. Also, in [1] we have used four cache configurations. Since the simulation results did not show much difference between these various configurations, in this paper we use only one cache configuration - the 'middle' one from [1].

The simulated DDR3 SDRAM memory has the most recent characteristics: there are 8 banks per chip, 8 K rows per bank, row capacity is 2 KB, $T_{rp} = T_{ra} = T_{ca} = 12.5$ ns. The DRAM operating frequency is 800 MHz.

We have simulated executions of 11 benchmark programs: the 6 SPEC95 programs from [1] (*cc1*, *compress*, *ijpeg*, *li*, *m88ksim*, *perl*), *go* from SPEC95, *bzip2*, *gcc*, *mcf* from SPEC2000, and *anagram*. The last one finds all the possible words obtained by combinations of the letters of 3 names: Todd Austin, Scott Breach and Guri Sohi. For 5 of the programs (*compress*, *li*, *m88ksim*, *go*, *anagram*) the simulated L2 cache size of 2 MB was large enough for the entire program's data to be placed in it. As a consequence the DRAM had small number of accesses. For these programs we had also tried versions with smaller L2 cache sizes, which yielded larger numbers of DRAM accesses. To differentiate between these 2 versions, the versions with larger L2 caches are signed with a star (*) after the benchmark name. The new versions give variety to the benchmark program set, since they obtain different open row hit probabilities.

3. Dead Time Predictor

In this paper we use the same two predictors as in [1]: the close page predictor and the open page predictor. The first one also consists of two predictors: the dead time predictor and the zero live time predictor.

We used the same 4 variants for the dead time predictor as in [1]. In this paper we just test their performances on DDR3 SDRAM without introducing any improvements. We sign them as *com2*, *com4*, *sep2* and *sep4*. (com/sep stands for 'using one common register'/'using separate registers for each bank' for storing the access interval time and 2/4 stands for multiplying by 2/4.)

The 4 variants yield similar performances, with *com2* and *sep2* being slightly better than *com4* and *sep4*, as can be seen from Figs. 1 and 2. These figures show the average DRAM latencies expressed as processor clock cycles when using the open page policy, the 4 dead time predictors, and an ideal close page predictor (a predictor with 100% accuracy). It can be seen that for most of the programs there are improvements, since the average latency is less than the latency obtained by the open page policy. It can be also seen that for most of the programs the latency is still far from the latency that would be obtained by the ideal close page predictor. The zero live time predictor improves this.

Fig. 1 Aver. latencies (first 8 programs) for dead time pred.

Fig. 2 Aver. latencies (last 8 programs) for dead time pred.

4. Zero Live Time Predictor

In this paper we consider the three variants of zero live time predictors from [1] (*1b*, *2b*, *2br*) and also introduce a new one. They are added to *sep2*, being the best dead time predictor. Since *1b* uses one bit for each row in the system, while *2b* and *2br* uses two bits for each row in the system, the implementation is simple, in a form of a SRAM memory with suitable organization. However, DRAM capacity involved in a computer system is constantly increasing, so the total number of rows in the system may be great. For the simulated DRAM system capacity of 2 GB with all the other parameters as in Sect. 2, the total number of rows in the system is 128 K. That means we need 128 Kb or 16 KB of SRAM for *1b* and 256 Kb or 32 KB of SRAM for *2b* and *2br*.

The amount of 32 KB of SRAM can be reduced if we group adjacent rows. Namely, if a zero live time occurs at a particular row, then in case of good program locality, there is a probability that zero live times will also occur at rows near by that row. Since the *2b* predictor was the best of the 3 predictors mentioned above we have tried to group several adjacent rows so that they all have one common *2b* predictor. By grouping each set of 16 adjacent rows with one common *2b* predictor, as shown in Fig. 3, we have decreased the requisite SRAM to only 2 KB. Left part of Fig. 3 shows the old version of the zero live time predictor. For n rows in the system there are n predictors. The right part shows the new version, where 16 adjacent rows share one common predictor. Hence the number of the predictors in this case is $n/16$. The sign used for the new version in this paper is *2b16*. The obvious advantage of the new version is several times less hardware needed for implementation. Its disadvantage is

a possible decrease of prediction accuracy. In general, by sharing a single predictor among several rows the predictor's accuracy should decrease. This, however, does not have to happen in case of programs with good access locality, which was our motive for introducing this predictor.

Figures 4 and 5 show the average latency when using the zero live time predictor (the 4 variants). These figures also show the latencies when using the open page policy, the best variant of the dead time predictor (*sep2*), and the ideal close page predictor. We can see from Fig. 4 that adding the zero live time predictor significantly reduces the latency for the first 5 programs, compared to the latency obtained with the dead time predictor only. In these cases the latencies are also very close to the ideal ones. The last three programs do not have significant improvements and are not so close to the ideal latencies. However in these programs there are still improvements in using the zero live time predictors compared to using only *sep2*.

Fig. 3 New version (right) of the zero live time predictor.

Fig. 4 Aver. latencies (first 8 programs) for zero live time pred.

Fig. 5 Aver. latencies (last 8 programs) for zero live time pred.

From the last 8 programs (Fig. 5) only *anagram* has significant latency reduction and relative approach to the ideal latency. For the rest of the programs we mostly have preservation of the latency gained by the dead time predictor with some small exacerbations or small improvements. This is a consequence of the general decrease of the number of zero live times. In these cases the predictors could not do better, especially for the last 5-6 programs. We can conclude this by the fact that the obtained latencies are close to the ideal latencies.

If we compare the latencies among the different variants, *2b* and *2b16* are the best in all the cases, and *2br* joins them in some of the cases. Although one would expect *2b16* to be worse than *2b*, in all of the cases *2b16* yields lower or equal latencies than *2b*. The explanation lies in the fact that *2b* must 'spend' 2 zero live times for each row before starting to predict the live time as zero. Since *2b16* is shared between 16 rows it needs 2 zero live times for any of the 16 rows, after which it predicts the live time as zero for any of the 16 rows. This additionally justifies sharing the predictor among several adjacent rows.

5. Open Page Predictor

We amplified the close page predictor *2b16* with an open page predictor, with a same structure as in [1]. It consists of two tables: Row History Table (RHT) and Pattern History Table (PHT), as shown in Fig. 6. RHT stores the last k rows that were activated in each of the banks, so there are k fields in an item for each of the banks. PHT contains the predictions. Its size is significantly larger than RHT's. As such, PHT determines the amount of SRAM memory needed for the open page predictor. It has $m \leq n$ items, where n is number of bank rows. Each item contains j two-part fields: row and next predicted row. PHT access index is obtained as t least significant bits of the sum (truncated addition) of the last k row indexes from the proper item for that bank in RHT, so $m = 2^t$.

For $k = 4$, $m = 4096$, $j = 2$ and the adopted DDR3 SDRAM structure, implementation of the open page predictor would demand only 832b for RHT and 26 KB for PHT.

In order to decrease the significant size of PHT, we tried with $m = 1024$ (number of PHT items), which decreases PHT to only 6.5 KB. In this paper we sign these two versions for $m = 4096$ and $m = 1024$ as *m4K* and *m1K*.

Fig. 6 Structure of the open page predictor.

Fig. 7 Aver. latencies (first 8 programs) for open page pred.

Fig. 8 Aver. latencies (last 8 programs) for open page pred.

respectively, and add them to *2b16*. Decreasing the size of PHT can obviously lead to loss of performances. Some predictions that were written in PHT at one moment of time may have to be removed in the future for the sake of writing newer predictions. In cases where these removed predictions are needed in the future, there will be situations in which the open page predictor will have no predictions, thus missing opportunities to decrease the DRAM latency. This would not happen in case of a larger PHT table. On the other hand, PHT also may contain some obsolete predictions, which (if applied) would increase the DRAM latency. If these obsolete predictions are removed, it is possible a smaller table to yield better performances than a larger one.

The average latencies obtained when using the open page controller policy, the close page predictor *2b16* and the open page predictors *m4K* and *m1K*, are shown in Figs. 7 and 8. It can be seen that there are almost no differences between *m4K* and *m1K*, the only program where *m4K* is (only slightly) better than *m1K* is *mcf*. These results, which justify the decrease of the PHT size, also show that the complete predictor could be implemented in the near future. Namely, the total hardware requirements for all three predictors would be about 10 KB, and today's processors have 2 or 3 levels of caches with total capacity of several hundreds of KB. Another important thing can be noticed from Figs. 7 and 8. If close page autoprecharge controller policy was used then the latency would be 80 cycles ($T_{ra} + T_{ca}$) for all the programs, which is larger than the obtained latencies. This means that in all 16 cases we have obtained latencies

that are smaller than the latencies obtained by both open page and close page autoprecharge controller policy. The complete predictor based on *sep2*, *2b16* and *m1K* yielded latency reductions of 23.5% compared to open page policy and 24.1% compared to close page autoprecharge policy (these are average values for all 16 cases).

6. Conclusion

In this paper we explored possibilities of trying the same techniques from our previous paper [1] on contemporary DDR3 SDRAM memory, with additional improvements of the predictors. Since DDR2 SDRAM has practically twice the bandwidth that DDR SDRAM has, and DDR3 SDRAM has twice the bandwidth that DDR2 SDRAM has, the latency participation in the total time needed for transferring the data block from/to main memory to/from the last level of cache memory is constantly increasing with newer types of SDRAMs. This latency participation is about 70%-90% for DDR3 SDRAM memories. Besides trying all the predictors from [1] on DDR3 SDRAM, we tried a modification of the best zero live time predictor from [1]. The new version preserves the performance improvements and requires 16 times less hardware. We also tried a smaller version of the open page predictor which requires 4 times less hardware and yields the same performance improvements. The obtained complete predictor significantly reduces the average DRAM latency compared to both open page controller policy and close page autoprecharge controller policy and requires less than 10 KB of SRAM memory for a 2 GB DRAM memory system.

References

- [1] V. Stankovic and N. Milenkovic, "DRAM controller with a complete predictor," IEICE Trans. Inf. & Syst., vol.E92-D, no.4, pp.584-593, April 2009.
- [2] Elpida, User's Manual: New Features of DDR3 SDRAM, Document No.E1503E10 (Ver.1.0), www.elpida.com
- [3] Micron, 1 Gb DDR3 SDRAM MT41J128M4/8/16, Micron Technology, 2006.
- [4] D. Burger and T.M. Austin, "The SimpleScalar tool set, version 2.0," University of Wisconsin-Madison Computer Sciences Department Technical Report #1342, June 1997.
- [5] Z. Hu, S. Kaxiras, and M. Martonosi, "Timekeeping in the memory system: Predicting and optimizing memory behavior," 2003 IEEE ISSCC, Feb. 2003.
- [6] Y. Xu, A.S. Agarwal, and B.T. Davis, "Prediction in dynamic SDRAM controller policies," SAMOS 2009, ed. K. Bertels, N. Dimopoulos, C. Silvano, and S. Wong, LNCS 5657, pp.128-138, 2009.