Prediction Based DRAM Row-Buffer Management in the Many-Core Era

Manu Awasthi, David W. Nellans, Rajeev Balasubramonian, Al Davis
School of Computing, University of Utah

Abstract

Modern processors are experiencing interleaved memory access streams from different threads/cores, reducing the spatial locality that is seen at the memory controller, making the combined stream appear increasingly random. Traditional methods for exploiting locality at the DRAM level, such as open-page and timer-based policies, become less effective as the number of threads accessing memory increases. Employing closed-page policies in such systems can improve performance but it eliminates any possibility of exploiting locality.

In this paper, we build upon the key insight that a history-based predictor that tracks the number of accesses to a given DRAM page is a much better indicator of DRAM locality than timer based policies. We extend prior work to propose a simple Access Based Predictor (ABP) *that tracks limited access history at the page level to determine page closure decisions, and does so with much smaller storage overhead than previously proposed policies. We show that ABP, with additional optimizations, can improve system throughput by 12.3% and 21.6% over open and closed-page policies, respectively. The proposed ABP requires 20 KB of storage overhead and is outside the critical path of memory access.*

***Keywords*-DRAM row-buffer managagement; predictor;**

I. Introduction

The memory system continues to be a key performance bottleneck for modern multi-cores [1]. Each independent thread or process produces a stream of references that accesses a shared DRAM sub-system. While each of thee streams may contain some spatio-temporal locality, they are multiplexed into a single command stream at the memory controller. This interleaving of memory streams increases the randomness of accesses seen at DRAM.

The loss of locality due to intervening accesses defeats several traditional DRAM performance and power optimizations that were very effective in single-core systems, e.g. *open-page policy*. When the CPU requests a cache line, the DRAM chips read out a DRAM-page (typically 4 or 8 KB) into a row-buffer. If a subsequent request for another cache line is to the same DRAM-page, then the access can be serviced by the low latency row-buffer. If a request is made to a different DRAM-page (a *row-buffer conflict*), the existing row-buffer contents must be written back to the DRAM arrays before reading out the new DRAM-page into the row-buffer. To avoid the write-back latency, some systems adopt a *closed-page* policy where the open DRAM-page is written back to the DRAM arrays immediately after servicing a single access. Closed-page works well for random accesses but incurs higher power and delay overhead when some locality is present.

This work was supported in parts by NSF grants CCF-0811249, CCF-0916436, NSF CAREER award CCF-0545959, HP, Intel, SRC grant 1847.001, and the University of Utah.

II. Motivation and Proposal

A number of *hybrid* row-buffer management mechanisms covering the wide spectrum of possibilities between open-page and closed-page policies have been studied previously. So far, all the hybrid row-buffer schemes proposed [2–5], have mostly used *time* as the underlying mechanism to determine when to close a row-buffer. The policies differ in terms of (i) the granularity at which the timer data is maintained – per-bank or global, and (ii) the mechanism adopted to learn from past history. For the majority of proposals, the timer values are changed by a fixed amount depending on whether or not correct decisions were made by the previous timer values. Correct decisions which increase locality/row-buffer utilization are rewarded by keeping the row-buffer open longer, while damage control for wrong decisions is done by decreasing timer values.

The other approach to row-buffer management has been via prediction based policies. A number of previous studies [6] have explored predicting the length of time for which a row-buffer should be kept open [7]. Others have proposed using number of accesses to a row-buffer as the deciding mechanism for the duration of time that a row-buffer should be kept open.

The most relevant study in this regard was done by Xu et al. [8]. They propose using a two-level access based predictor, similar to a branch predictor [9]. They propose a large family of predictors each differentiated by the granularity at which prediction meta-data is tracked – per-page, per-bank or global. The authors acknowledge that maintaining information at per-page level, although is the best organization, has very high overheads for practical implementations.

Some other studies have also used predictor based policies [6, 7], but except for [8], no one has on a per

DRAM-page granularity. This has been done to avoid the excessive hardware overheads associated with maintaining meta-data at a fine scale. Furthermore, no prior work has evaluated the efficacy of proposed policies in a multi-core setting.We believe that in a multi-core setting, *access* based methods will prove to be more effective.

The intuition behind using access based prediction is that perfect predictions have performance equivalent to an oracular closure policy. Closing the DRAM-page immediately after a correct prediction results in the minimal possible performance conflicts. Contrast this to a timer based policy, where we must predict a timer value N exactly across hundreds or thousands of cycles to obtain the same oracular performance.

As mentioned before, tracking histories at a per DRAM-page granularity incurs a high overhead for two-level predictor implementation proposed in [8]. We propose ABP, which is a one-level, low cost implementation of a Xu et al. [8] like predictor scheme. The ABP-based row-buffer closure policy is implemented as follows: On first access to a DRAM-page, the predicted number of accesses is looked up in the history table. If no entry exists, the row-buffer is left open until a page-conflict occurs. Upon closure, the number of accesses in the history table that occurred for this DRAM-page are recorded. If an entry exists in the table, the DRAM-page is closed after the specified number of accesses or when a page-conflict occurs. If a page-conflict occurs, the number of accesses in the history table is decremented by 1, which hopefully will lead to a perfect prediction next time that DRAM-page is accessed. If we close the DRAM-page and a different DRAM-page is opened on the next access, our prediction was perfect and the value need not be updated. If we bring the same DRAM-page back into the row-buffer, it is allowed to remain open until a page-conflict happens. At this point the history table is updated with the aggregate number of accesses so that pre-mature closure is unlikely to happen again.

The history overhead is kept tolerable by simply caching the most recent predictions instead of maintaining an exhaustive list. The predictor is organized as a 2048-set/4-way cache, with each of the 32 DRAM banks having a 64-set 4-way predictor.

The Xu09 predictor collects row-hit and row-miss history in a n-bit shift register called the history register (HR). This history then indexes into a history table (HT) which keeps a saturating counter for all of possible 2^n combinations. The saturating counter value decides the row-buffer management policy for the next accesses. Based on the organization of choice, both the HR and the HT can be maintained at either global, per-bank, or per-page level. We model per-bank HRs and a global HT.

Look-ups to the predictor history table are not on the critical path as the prediction is required only after the DRAM-page has been opened (via a RAS) and read (via a CAS). This predictor table, co-located with the memory controller has an average hit rate of 92.6%. Aside from the prediction history table, these hardware requirements are the same as those required by per-bank timers.

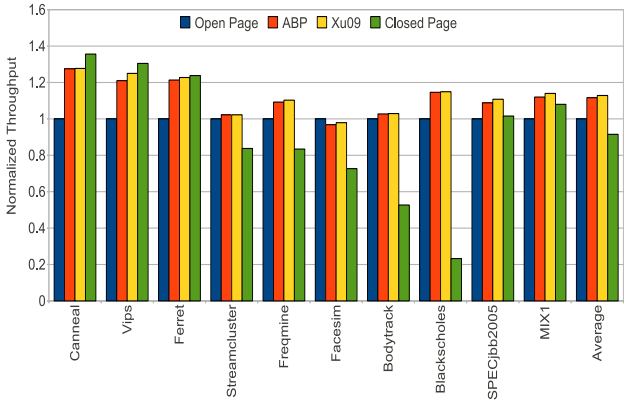

Figure 1. Access based prediction policies vs Open and Closed page

III. Conclusions

We evaluate previous work on access-based predictors (*Xu09* in Figure 1, [8]) and compare them to ABP, open- and closed-page policies. In a multi-core setting, access-based predictor mechanisms perform better than open- or closed-page policies. ABP can achieve the same performance as Xu09, with significantly smaller storage overhead, even though it operates on a page granularity. This is in contrast to Xu09. which has to maintain global information to reduce overheads. Given the strong performance gains, low variability across workloads, and a fraction of the overheads of similar performing access-based policies, we believe that ABP is a good candidate to replace the open-row policy in future many-core systems.

References

[1] B. Rogers *et al.*, "Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling," in *Proceedings of ISCA*, 2009.

[2] T. Rokicki, "Method and Computer System for Speculatively Closing Pages in Memory," 2002, United States Patent, Number 6389514-B1.

[3] O. Kahn and J. Wilcox, "Method for Dynamically Adjusting a Memory Page Closing Policy," 2004, United States Patent, Number 6799241-B2.

[4] B. Sander, P. Madrid, and G. Samus, "Dynamic Idle Counter Threshold Value for Use in Memory Paging Policy," 2005, United States Patent, Number 6976122-B1.

[5] S. Miura, K. Ayukawa, and T. Watanabe, "A Dynamic-SDRAM-Mode-Control Scheme for Low-Power Systems with a 32-bit RISC CPU," in *Proceedings of ISLPED*, 2001.

[6] S.-I. Park and I.-C. Park, "History-Based Memory Mode Prediction For Improving Memory Performance," in *Proceedings of ISCAS*, 2003.

[7] V. Stankovic and N. Milenkovic, "DRAM Controller with a Close-Page Predictor," in *Proceedings of EUROCON*, 2005.

[8] Y. Xu, A. Agarwal, and B. Davis, "Prediction in dynamic sdram controller policies," in *In Proceedings of SAMOS*, 2009.

[9] T.-Y. Yeh and Y. N. Patt, "Alternative implementations of two-level adaptive branch prediction," pp. 124–134, 1992.