Cache Coherence Enabled Adaptive Refresh for Volatile STT-RAM

Jianhua Li[1,2], Liang Shi[1,2], Qing'an Li[1,3], Chun Jason Xue[1], Yiran Chen[4], Yinlong Xu[2]
[1] Department of Computer Science, City University of Hong Kong, Hong Kong.
[2] College of Computer Science & Technology, University of Science & Technology of China, China.
[3] College of Computer Science, Wuhan University, Wuhan, China.
[4] Department of Electrical and Computer Engineering, University of Pittsburgh.
{jianhual,shil0704}@mail.ustc.edu.cn, ww345ww@gmail.com, jasonxue@cityu.edu.hk, yic52@pitt.edu, ylxu@ustc.edu.cn

***Abstract*—Spin-Transfer Torque RAM (STT-RAM) is extensively studied in recent years. Recent work proposed to improve the write performance of STT-RAM through relaxing the retention time of STT-RAM cell, magnetic tunnel junction (MTJ). Unfortunately, frequent refresh operations of volatile STT-RAM could dissipate significantly extra energy. In addition, refresh operations can severely conflict with normal read/write operations and results in degraded cache performance. This paper proposes Cache Coherence Enabled Adaptive Refresh (CCear) to minimize refresh operations for volatile STT-RAM. Through novel modifications to cache coherence protocol, CCear can effectively minimize the number of refresh operations on volatile STT-RAM. Full-system simulation results show that CCear approaches the performance of the ideal refresh policy with negligible overhead.**

I. Introduction

STT-RAM [3], [15] is a promising candidate for conventional SRAM caches given its attractive features, negligible leakage power, high storage density and fast access speed. However, the high power consumption and long latency write operations [4], [7], [12] impede the widespread adoption of STT-RAM. To tackle this issue, recent work [10] proposed to design high write performance STT-RAM caches through reducing the retention time of STT-RAM cell. Under 10 F^2 MTJ size, a 56 μs retention time STT-RAM with optimal read and write performance can be obtained [2]. To prevent potential data loss due to the reduced retention time, a simple DRAM-style refresh policy [10] was utilized to periodically refresh the cache blocks.

The requirement of cache block refresh will impede the adoption of volatile STT-RAM as the large last-level cache (LLC) for multicore systems. Figure 1 shows the performance comparison for a 16-core system with 8MB volatile STT-RAM LLC [1], configured with DRAM-style refresh policy [10] and an ideal refresh policy when running several LLC-intensive PARSEC workloads [1]. *Ideal refresh* assumes the refresh operation does not conflict with normal accesses and the refresh operations does not consume extra power. As shown in Figure 1, due to the conflict between refreshes and normal accesses, the average IPC of *facesim* is reduced by 9.4%. In addition, the refresh operation also induces extra power dissipation, for instance, the LLC power dissipation is increased by 17.4% for *facesim*.

[1]The detailed system setup is presented in Section IV-A.

978-3-9815370-0-0/DATE13/©2013 EDAA

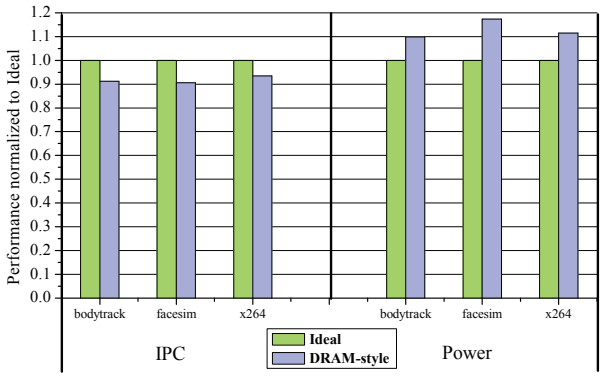

Fig. 1. Comparison of DRAM style refresh and Ideal refresh

In this work, we propose Cache Coherence Enabled Adaptive Refresh (CCear) to minimize the number of refresh operations in volatile STT-RAM LLC. Specifically, CCear interacts with a modified cache coherence protocol to reduce the number of refresh operations. The main contribution of this paper is as follows:

- Presents an analysis of the performance impact caused by the refresh operations on volatile STT-RAM LLC.
- Proposes Cache Coherence Enabled Adaptive Refresh (CCear) policy to minimize the number of refresh operations for shared LLC blocks.
- Through full-system simulation, we show that volatile STT-RAM with CCear yields comparable performance with an ideal refresh-off volatile STT-RAM.

The remainder of this paper is organized as follows. The related work is presented in Section II. The proposed CCear Policy is presented in Section III, while the experiments and performance analysis are demonstrated in Section IV. Finally, Section V concludes the paper.

II. Related Work

Recent work [10] proposed to relax the retention time of STT-RAM cells through shrinking the MTJ planar area.

The resultant STT-RAM has significant write performance improvement compared with its non-volatile counterpart. In addition, a DRAM-style refresh policy was integrated in the relaxed volatile STT-RAM [10] to refresh the cells before the ending of retention time.

Sun et al. [13] proposed to further relax the retention time of STT-RAM cells and evaluated the performance of the resultant STT-RAM as L1 caches. For ultra-low retention time STT-RAM caches, the cells have to be frequently refreshed wherein the refresh can conflict with normal read and write accesses and consume extra power. To tackle this issue, a counter based refresh scheme is proposed in [13] to mitigate the overhead of refresh. Each cache block needs a counter to indicate the lifetime of the block and the counter should be frequently updated by a global clock.

Jog et al. [5] found that the average interval between two successive writes to the same L2 cache block is around 10 ms. Based on this observation, a 10 ms retention time STT-RAM is designed and evaluated as the L2 cache. Different from [10], [13], the expiring blocks are dynamically written to a small buffer.

Different from the previous work, CCear is the first work to exploit the coherence information of cache blocks to mitigate the refresh overhead of volatile STT-RAM. The implementation of CCear is simple and the storage overhead is negligible. Moreover, CCear can be applied to volatile STT-RAM with ultra low retention time.

III. Adaptive Refresh Policy

In this section, we first illustrate the architecture of multi-core systems which integrates volatile STT-RAM as the LLC. Subsequently, the proposed CCear policy is presented.

A. System Architecture

Tiled architecture [14] is preferred for future multicore systems because its simplicity and scalability, wherein each tile consists of a core, private L1 I/D caches and a shared L2 slice. In this work, we assume that the metadata of L2 cache including tag and directory information are stored in SRAM wherein the L2 data array is implemented using volatile STT-RAM [10]. Differ from data array, the tag array and the directory are rarely written even for write-intensive workloads. Therefore, utilizing SRAM to store the tag and directory information can eliminate the refresh operations which are power-consuming. More important is that SRAM based tag and directory information are beneficial to the proposed Adaptive Refresh Policy which will be introduced in the following sections.

B. Cache Coherence Enabled Adaptive Refresh

In CCear, the coherence state of L2 blocks is exploited to minimize the number of refresh operations of the volatile STT-RAM. Figure 2 depicts the refresh policy in CCear. Under CCear, each shared block will be refreshed for N times after loading from main memory or writing back from L1 cache. Specifically, each cache block is associated with a counter, *ref_counter*, to indicate that how many times this block has been refreshed. Initially, the *ref_counter* counter for each shared LLC block is set to N. N is a user defined parameter and larger N indicates more data request will directly obtain the target block at L2 cache. How to obtain the optimal N is workload-specific and out of the scope of this paper.

```
1:  if block(addr) is shared
2:     if block(addr).valid_bit ==1
3:        if block(addr).ref_counter > 1
4:           refresh(block(addr));
5:           block(addr).ref_counter--;
6:        else
7:           block(addr).valid_bit = 0;
8:           return;
9:     else
10:       return;
    // end  if block(addr) is shared
```

Fig. 2. **CCear Refresh Policy**

As shown in Figure 2, when a refresh operation is activated on a shared block, the *valid_bit* of the block is first checked as indicated by line 2. If the block is valid, then the $ref\_counter$ is checked to see whether the value is bigger than 1. If the block has not been refreshed by N times, CCear will refresh the block and reduce the counter by one (line 3∼5). Otherwise, CCear will set the *valid_bit* to 0 which indicates that the block is not refreshed and the data in the block will be lost soon. CCear will do nothing if the block is not valid which signifies that the block is not refreshed previously and the data in the block has vanished.

In the proposed refresh scheme, the data in LLC could be vanished because each block will be only refreshed for fixed times. The vanished data will make the normal coherence protocol unable to ensure the consistency in cache hierarchies. In other words, subsequent requests to the vanished STT-RAM block can not get the correct data using the conventional cache coherence protocol. For the purpose of maintain the cache consistency in CCear, the following modifications to conventional MESI coherence protocol are proposed.

- A novel coherence state W is added to the L1 cache blocks. An L1 block with W state indicates that one or multiple replicas of the block is also present in other L1 caches. L1 block in W state is not allowed to be silently evicted. Upon replacing one L1 block in W state, the block is explicitly written back to allow the L2 cache to obtain the correct copy in case the data has vanished.
- On account of the addition of W state, additional coherence state transitions are added between W state and other states as indicated by the solid arrows in Figure 3.
- The state transitions, $M \rightarrow S$ and $E \rightarrow S$, are removed from conventional $MESI$ protocol [11]. In addition, the L1 block in I state will be translated to S state only if the

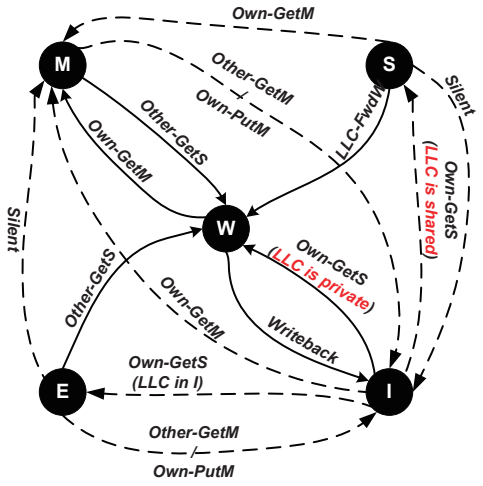

Fig. 3. Coherence state transition for L1 cache

block has one or multiple replicas in other L1 caches as depicted in Figure 3. If there are no replica in L1 caches (privately owned by L2 cache), the state will be translated to W state.

- The L1 data request will be served with three hop coherence mechanism based on the directory information stored in SRAM if the target L2 data has vanished. Note that the tag and directory information stored in SRAM will ensure the correct cache coherence operations even though the data disappears.

The dash arrows in Figure 3 represent the state transitions in conventional $MESI$ protocol while the solid arrows indicate the state transitions between W state and other states. Note that the added W state is different from O state in conventional MOESI protocol. For an L1 block in W state, the shared block in low level memory is clean while for an L1 block in O state the corresponding shared block could be dirty. Under CCear, each write back from L1 cache will reset the $valid\_bit$ of the corresponding L2 block to 1 in addition to the coherence state transition. Through such write back reset operation, the subsequent requests to this block before the next refresh operation to it can be directly obtained from the L2 cache without three-hop operations. Note that the W state is different from the O state in MOESI protocol which indicates the target block is modified.

The objective of adding the W state is to make sure that the L2 block with vanished data will eventually receive the correct data through writing back the L1 replica in W state. In conventional $MESI$ protocol, the L1 block in S state is allowed to be silently evicted without noticing L2 cache. If all the replicas in L1 caches are evicted silently, the L2 cache will not receive the correct data permanently and the corresponding L2 block will be in a fault state. CCear only allows one L1 block in W state. When an L1 block in W state is written back to L2, the L2 cache controller will forward a control message (indicated with $LLC - FwdW$) to the nearest L1 replica if there still exists replica of this block in L1 caches. Upon receiving the control message, the corresponding L1 cache controller will switch the state of the replica to W state as shown in Figure 3.

TABLE I
MAIN SIMULATION PARAMETERS

Parameter	Value
Processor	16 UltraSPARC III+ in-order cores 2GHz, Operating System: Solaris 10
L1 I/D Cache	Private, 32/32KB, 2-way, 2-cycle latency 64 bytes block size, write-back
Last-Level Cache	8MB STT-RAM, 16 banks 32-way, 64B block, write-back
Coherence mechanism	MESI directory protocol
Main memory latency	200 cycles
Interconnect	4×4 mesh, 128-bit links, 1-cycle link latency 2-cycle router latency.

TABLE II
CHARACTERISTICS OF LLC BANK

Read latency	Write latency	Read energy	Write energy	Leakage power	Area
0.73 ns	2.21 ns	0.3 nJ	1.65 nJ	78 mW	1.2 mm^2

IV. Experiments and Analysis

In this section, we first present the experimental setup. Subsequently, the experimental results and the corresponding analysis will be presented. Finally, we present the implementation overhead of CCear.

A. Experimental Setup

We model a 2GHz chip multicore processor with 16 in-order cores using Simics [8]. Each core is configured with 32KB private instruction/data cache and a shared 512KB L2 slice. We modified the MESI directory protocol in GEMS framework [9] to support CCear. We assume the main memory has a fixed latency of 200 cycles. The L2 data array is implemented with the 56μs retention time STT-RAM [10]. The memory subsystem is simulated with GEMS [9] and the on-chip network power dissipation is calculated with Orion [6]. The details of our simulator parameters are shown in Table I and II.

We compared the proposed CCear scheme with the DRAM-style refresh policy used in [10]. We utilize energy dissipation and instruction per cycle (IPC) as the performance metrics for the evaluated schemes. For CCear, the refresh times is set to four. We simulated a set of PARSEC applications [1] with different intensity of read/write operations. All the workloads are executed with *simmedium* input sets. We evaluated the whole parallel region, known as region of interest (ROI), of the selected applications.

B. Results and Analysis

1) Power: Figure 4 presents the power dissipation for the evaluated schemes including the on-chip network power. The always refresh mechanism makes DRAM-style refresh policy consume much more power compared with the ideal policy. For *swaption*, the actual power dissipation of the volatile STT-RAM LLC is significantly lower than the power consumed in other workload due to the fewer accesses to LLC which

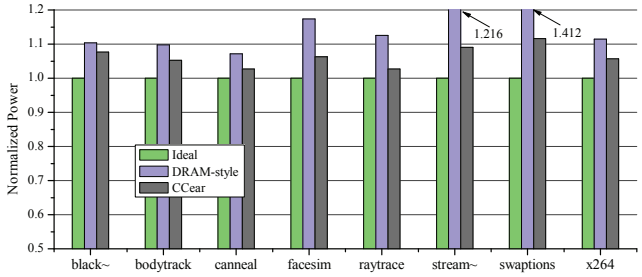

Fig. 4. Impact on power dissipation

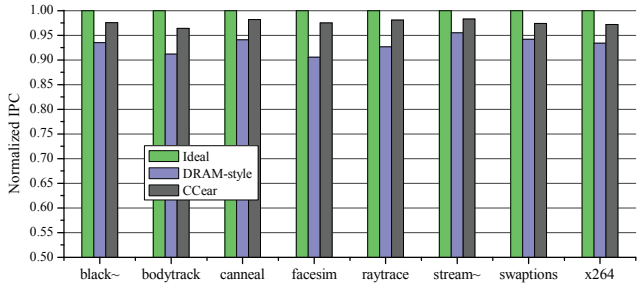

Fig. 5. Impact on instruction per cycle (IPC)

also represents little on-chip routing power. As a result, the DRAM-style refresh policy increases the power consumption by 41.2% compared with the baseline ideal policy as indicated in Figure 4.

By comparison, CCear outperforms the DRAM-style refresh policy for all of the evaluated workloads. CCear reduces the power dissipation by 10.1% on average and up to 29.6% compared with the DRAM-style refresh policy taking the ideal refresh policy as the baseline. The main contributor to the power yield of CCear is the reduced number of refreshes to the shared blocks.

2) IPC: Figure 5 shows the normalized IPC for the evaluated schemes. As indicated in Figure 5, CCear outperforms the DRAM-style refresh policy for all workloads. Taking the ideal refresh policy for reference, CCear improves the IPC metric by 2.8% to 6.9% compared with DRAM-style refresh policy. The main contribution to the performance improvement comes from the reduced conflicts between refresh operations and normal cache accesses. This can be confirmed through checking the performance of memory intensive workloads. Due to the conflicts, the IPC of DRAM-style policy is reduced by 6.8% on average compared with the ideal policy.

C. CCear Overhead Analysis

For CCear, each LLC block needs to be associated with an extra bit to indicate whether the block is expired or not. For 64B block size, the storage overhead is less than 0.2%. In addition, each LLC block needs a counter to indicate the number of block refresh times. For the simulated 16-tile multicore system setup, the total storage overhead including the valid bit is less than 1%, wherein each counter needs 4 bits. Compared to the large capacity LLC, the storage overhead of CCear can be assumed to be negligible.

V. Conclusion

In this paper, we propose an adaptive refresh policy called Cache Coherence Enable Adaptive Refresh (CCear) for volatile STT-RAM last-level caches. Through novel modifications to cache coherence protocol, CCear can adaptively and effectively minimize the number of refresh operations of volatile STT-RAM LLC. Our results of full-system simulation show that CCear can effectively reduce the refresh operations to improve the performance of volatile STT-RAM caches. Specifically, CCear outperforms the DRAM-style refresh policy for all the studied workloads.

VI. Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable feedback and improvements to this paper. This work is supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 123811].

References

[1] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[2] E. Chen, D. Apalkov, et al. Advances and Future Prospects of Spin-Transfer Torque Random Access Memory. *IEEE Transactions on Magnetics*, 46(6):1873–1878, 2010.

[3] M. Hosomi, H. Yamagishi, et al. A Novel Nonvolatile Memory with Spin Torque Transfer Magnetization Switching: Spin-RAM. In *International Electron Devices Meeting (IEDM '05)*, pages 459–462, 2005.

[4] J. Hu, C. J. Xue, et al. Reducing Write Activities on Non-Volatile Memories in Embedded CMPs via Data Migration and Recomputation. In *Annual Design Automation Conference (DAC '10)*, pages 350–355, 2010.

[5] A. Jog, A. K. Mishra, et al. Cache Revive: Architecting Volatile STT-RAM Caches for Enhanced Performance in CMPs. In *Annual Design Automation Conference (DAC '12)*, pages 243–252, 2012.

[6] A. Kahng, B. Li, et al. Orion 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. In *Design, Automation Test in Europe (DATE '09)*, pages 423–428, 2009.

[7] J. Li, C. Xue, and Y. Xu. STT-RAM based Energy-Efficiency Hybrid Cache for CMPs. In *International Conference on Very Large Scale Integration (VLSI-SoC '11)*, pages 31–36, 2011.

[8] P. Magnusson, M. Christensson, et al. Simics: A Full System Simulation Platform. *Computer*, 35(2):50–58, feb 2002.

[9] M. M. K. Martin, D. J. Sorin, et al. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) toolset. *SIGARCH Computer Architecture News*, 33:92–99, 2005.

[10] C. Smullen, V. Mohan, et al. Relaxing Non-Volatility for Fast and Energy-Efficient STT-RAM Caches. In *International Symposium on High Performance Computer Architecture (HPCA '11)*, pages 50–61, 2011.

[11] D. J. Sorin, M. D. Hill, and D. A. Wood. *A Primer on Memory Consistency and Cache Coherence*. Synthesis Lectures on Computer Architecture. Morgan and Claypool Publishers, May 2011.

[12] J. Li, L. Shi, C. Xue, C. Yang and Y. Xu. Exploiting Set-Level Write Non-Uniformity for Energy-Efficient NVM-based Hybrid Cache. In *IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia '11)*, pages 19–28, 2011.

[13] Z. Sun, X. Bi, et al. Multi Retention Level STT-RAM Cache Designs with A Dynamic Refresh Scheme. In *IEEE/ACM International Symposium on Microarchitecture (MICRO '11)*, pages 329–338, 2011.

[14] M. B. Taylor, J. Kim, et al. The RAW Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs. *IEEE Micro*, 22(2):25–35, Mar. 2002.

[15] C. J. Xue, Y. Zhang, et al. Emerging Non-Volatile Memories: Opportunities and Challenges. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '11)*, pages 325–334, 2011.