Preventing STT-RAM Last-Level Caches from Port Obstruction

JUE WANG, Pennsylvania State University

XIANGYU DONG, Qualcomm Technology, Inc.

YUAN XIE, Pennsylvania State University

Many new nonvolatile memory (NVM) technologies have been heavily studied to replace the power-hungry SRAM/DRAM-based memory hierarchy in today's computers. Among various emerging NVM technologies, Spin-Transfer Torque RAM (STT-RAM) has many benefits, such as fast read latency, low leakage power, and high density, making it a promising candidate for last-level caches (LLCs).¹ However, STT-RAM write operation is expensive. In particular, a long STT-RAM cache write operation might obstruct other cache accesses and result in severe performance degradation. Consequently, how to mitigate STT-RAM write overhead is critical to the success of STT-RAM adoption. In this article, we propose an obstruction-aware cache management policy called OAP. OAP monitors cache traffic, detects LLC-obstructive processes, and differentiates the cache accesses from different processes. Our experiment on a four-core architecture with an 8MB STT-RAM L3 cache shows a 14% performance improvement and 64% energy reduction.

Categories and Subject Descriptors: B.3.2 [Memory Structures]: Design Styles, Cache Memories

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Nonvolatile memory, STT-RAM, last-level caches, port obstruction, performance improvement

ACM Reference Format:

Jue Wang, Xiangyu Dong, and Yuan Xie. 2014. Preventing STT-RAM last-level caches from port obstruction. ACM Trans. Architect. Code Optim. 11, 3, Article 23 (July 2014), 19 pages.

DOI: <http://dx.doi.org/10.1145/2633046>

1. INTRODUCTION

While Moore's law is still offering smaller and faster transistors, the rate of improvement in microprocessor speed has exceeded the one in memory speed. The integration of multiple cores on a single die accentuates the already daunting *memory-wall* problem.

¹In this work, we assume a 3-level cache hierarchy, and the terms "L3 cache" and "last-level cache" are used interchangeably.

Extension of Conference Paper: This submission is extended from OAP: An Obstruction-Aware Cache Management Policy for STT-RAM Last-Level Caches, published on the Conference on Design, Automation and Test in Europe 2013. The additional material provided in the submission includes more detailed motivation examples and explanations, the Static Threshold OAP architecture and its disadvantages, traffic overhead analysis, OAP sensitivity studies on cache write latency, cache bank number, system core number, and period length.

This work is supported in part by SRC grants, NSF 1218867, 1213052, 0903432, and by the DoE under Award Number DE-SC0005026.

Authors' addresses: J. Wang and Y. Xie, Computer Science and Engineering Department, Pennsylvania State University; emails: jzw175@cse.psu.edu; yuanxie@cse.psu.edu; X. Dong, Qualcomm Technology, Inc; email: xydong@cse.psu.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2014 ACM 1544-3566/2014/07-ART23 \$15.00

DOI: <http://dx.doi.org/10.1145/2633046>

The memory hierarchy design has to catch up and tremendously scale in performance, energy efficiency, and storage capacity to sustain the processing demands of a wide variety of next-generation applications. To respond, deeper cache hierarchy and larger cache capacity have been observed in contemporary microprocessor designs. For example, the Intel i7-3930K processor is equipped with a 12MB SRAM L3 cache, and the IBM POWER7 processor has an unprecedented 32MB embedded DRAM (eDRAM) L3 cache. However, conventional SRAM and eDRAM are facing constraints of cell area and leakage energy consumption with technology scaling.

Recently, new memory technologies such as *Spin-Transfer Torque RAM (STT-RAM)*, *Phase-change RAM (PCRAM)*, and *Resistive RAM (ReRAM)* have started being explored as potential alternatives to SRAM and DRAM. While PCRAM and ReRAM have the write endurance issue, and hence are not suitable to be used as an on-chip cache, STT-RAM is an attractive option for last-level caches (LLCs) because of its fast read latency, high density, low leakage, and nonvolatility. For example, Toshiba revealed its plan in using STT-RAM to replace a 512KB SRAM L2 cache for low-power purpose [Nomura et al. 2011]. Another example is that replacing DRAM with STT-RAM in data centers can reduce power by up to 75% [Driskill-Smith 2011].

While incorporating STT-RAM cache has many advantages, unfortunately, one of the major disadvantages of STT-RAM is the latency and energy overhead associated with its write operations. In particular, a long STT-RAM write operation might obstruct other cache accesses [Dong et al. 2008; Sun et al. 2009], especially when multiple processes are running in parallel. This could cause a severe performance degradation.

Therefore, a mitigation technique to minimize the write overhead is required before any successful STT-RAM cache design. In this article, we propose a sophisticated cache management policy for STT-RAM LLCs. The key contributions can be summarized as follows:

- (1) We define a type of process characteristic called LLC obstructive. Processes with this characteristic can cause significant performance degradation in an STT-RAM LLC-based system, and it also negatively affects the performance of other processes running in parallel.
- (2) We design an obstruction-aware monitoring mechanism, OAM, which monitors LLC access traffic and detects LLC-obstructive processes during runtime.
- (3) We propose an obstruction-aware cache management, OAP, which differentiates the accesses generated by LLC-obstructive processes from others.

The proposed OAP scheme has negligible hardware overhead, and it can significantly improve the performance and reduce energy consumption for future multicore systems using STT-RAM LLCs. Our experiment results show that the performance of a projected four-core system with an 8MB STT-RAM LLC cache is improved by 14% on average and up to 42% after adopting OAP. In addition, the energy consumption is reduced by 64%. This demonstrates how our proposed OAP scheme is effective in mitigating the performance and energy overhead associated with STT-RAM cache write operations, paving the way for STT-RAM adoption in future microprocessor designs.

2. BACKGROUND AND MOTIVATION

In this section, we briefly review the background of the STT-RAM technology and use a motivational example to demonstrate why the port obstruction problem can cause huge performance loss.

2.1. STT-RAM Technology

The original MRAM (Magnetoresistive RAM) is Toggle-Mode MRAM [Maffitt et al. 2006], which relies on magnetic field-based switching. STT-RAM is the second

Fig. 1. The conceptual view of an STT-MRAM cell.

Table I. Characteristics of 8MB SRAM and STT-RAM Caches (32nm)

Memory type	SRAM	STT-RAM
Cell factor (F^2)	146	40
Read latency (ns)	11.1	11.4
Write latency (ns)	11.1	22.5
Read energy (pJ)	15.8	17.5
Write energy (pJ)	13	172
Leakage power (mW)	14.1	0.14
Area (mm^2)	11.5	3.16

generation of MRAM. The basic storage element in STT-RAM is a magnetic tunnel junction (MTJ) shown in Figure 1. Each MTJ is composed of two ferromagnetic layers; one has a fixed magnetization direction and the other has a free one. Since the free layer can change its direction, the relative direction of these two is used to represent a digital “0” or “1.” In STT-RAM technology, a switching current flowing from bitline to sourceline changes the direction of the MTJ free layer and turns the cell into “0” (parallel); a switching current flowing from sourceline to bitline turns the cell into “1” (antiparallel).

2.2. Using STT-RAM LLCs

Compared to conventional SRAM cache, the advantages of STT-RAM cache are smaller area and lower leakage power. The cell size of STT-RAM is currently in the range of $13F^2$ [Kawahara et al. 2007] to $100F^2$ [Tsuchida et al. 2010], where F is the feature size. This cell size is much smaller than the SRAM size (e.g., $146F^2$ [Thoziyoor et al. 2008]). The small cell size of STT-RAM can greatly reduce the silicon area occupied by caches. In addition, STT-RAM has zero leakage power consumption from memory cells due to its nonvolatility. Previous work [Kim et al. 2003] shows the leakage energy can be as high as 80% of total energy consumption for an L2 cache in a 130nm process. Thus, using STT-RAM last-level caches to eliminate standby leakage power is an attractive choice.

Table I shows the characteristics of a four-bank eight-way 8MB SRAM cache and its STT-RAM counterpart. The estimation is given by NVSim [Dong et al. 2012], a performance, energy, and area model based on CACTI [Thoziyoor et al. 2008]. It shows that the STT-RAM cache has a much smaller area and leakage power than the one of SRAM. However, as also shown in Table I, the write latency of STT-RAM is around two times longer than SRAM. This long write latency becomes a sensitive parameter in the STT-RAM L3 cache and causes the L3 cache port obstruction problem.

2.3. Motivation 1: Port Obstruction

As the last-level cache, L3 is usually implemented using a single-port memory bitcell due to the infeasible cost associated with multiport bitcell designs. For example, building a dual-port STT-RAM cell requires at least four transistors [Rao and Kim 2011]

Table II. Cache Hierarchy of Three Different Systems

System name	Cache hierarchy
2-LC	4-core, private SRAM 32KB L1, SRAM 256KB L2
3-LC-SRAM	2-LC + SRAM 8MB shared L3
3-LC-STTRAM	2-LC + STT-RAM 8MB shared L3

Fig. 2. Compared to the system 2-LC, the normalized IPC of 3-LC-SRAM and 3-LC-STTRAM using four-duplicate workloads.

(a four times larger cell layout area), which is not affordable in large-capacity L3 designs. For single-port STT-RAM caches, an ongoing write operation can cause a long L3 cache port obstruction and delay all the following read operations that are on the critical path from the performance aspect. When the number of pending read operations exceeds the number of MSHR entries of the requester (e.g., CPU or upper-level cache) [Kroft 1981], the system must stall. Because of this port obstruction, while some workloads can benefit from the larger capacity of STT-RAM LLCs, it is possible that the system performance is not improved but degraded when running some write-intensive workloads.

A common practice to hide write latency is to enlarge the write buffer size. Write buffer can hold the incoming writing data and fill them into the destination cache line only when the cache becomes idle or the write buffer reaches the high water mark. Write buffering is usually effective in traditional SRAM LLCs since SRAM write speed is sufficiently fast to drain all the write buffer entries during the cache idle period. However, write buffering becomes less effective for the STT-RAM cache as its write is too long to hide. In addition, the practical write buffer size is limited by design complexity and fully associative look-up overhead.

To quantify this write-induced port obstruction problem, we simulate three different systems as configured according to Table II (see Section 4 for details on our simulation methodology): one with only two levels of caches, one with an additional SRAM L3 cache, and one with an additional STT-RAM L3 cache. Figure 2 shows the normalized IPC comparison among these three systems. Compared to 2-LC, the performance of 3-LC-SRAM is improved by 3% on average, but the performance of 3-LC-STTRAM is degraded by 14% on average (up to 35%). While adding an STT-RAM L3 cache improves the performance for some workloads (e.g., *hmmer*), it might heavily degrade the performance for write-intensive workloads (e.g., *lbm*) as well.

2.4. Motivation 2: Process Interference

Besides the performance degradation caused by a workload's own write intensity, we must notice that the LLC is commonly a shared resource, and one process that obstructs the cache port can block the normal cache accesses generated by other concurrent processes in a multicore system.

Fig. 3. Normalized IPC of 3-LC-SRAM and 3-LC-STTRAM for two groups of four-mixed workloads, in which the first core run different workloads and the others are the same.

To illustrate this problem, we simulate four different workloads running simultaneously on a four-core system. We use two different mixed groups: *mix1* and *mix2*. Figure 3 shows the normalized IPC of each workload in these two mixed groups, and the names of running workloads are also listed.

In our experiment design, we assign four different workloads to Group 1 and copy the same configuration to Group 2 but intentionally replace the workload running on the first core with a write-intensive workload *cactusADM* in Group 2. The result shows that all the workload IPCs in Group 1 are only reduced by 1%. However, on the contrary, not only the IPC of the first core but also all of the IPCs in Group 2 are degraded significantly (10% to 25%). This demonstrates that the write-intensive process not only causes performance loss to itself but also interferes with the normal cache accesses initiated by other processes.

Based on these observations, we define a type of process characteristic called “LLC obstructive.” If a workload is LLC obstructive, the performance of itself can be degraded significantly by adding an STT-RAM LLC, and worse, it might also affect the performance of other workloads in a negative way.

Therefore, we need a new STT-RAM L3 cache management policy, and its task is to first identify any potential LLC-obstructive processes on the fly and then avoid the performance degradation caused by such LLC-obstructive processes.

3. OBSTRUCTION-AWARE CACHE MANAGEMENT POLICY

In this article, our basic concept is to enhance the existing cache management policy so that it can detect those LCC-obstructive processes, block their cache accesses, and reserve the LLC capacity for well-behaved processes. In this section, we describe how to implement such a policy.

3.1. A Naive Approach: Using Static Threshold

To design an obstruction-aware cache management policy, the most critical problem is how to find LLC-obstructive processes during runtime. We start from a relatively straightforward method called “static threshold obstruction-aware management policy” (SOAP), which uses a predetermined threshold to determine whether a process is LLC obstructive. Based on our motivational experiments in Section 2, we can observe that LLC-obstructive processes commonly have two features:

- (1) **High utilization:** The LLC port occupancy is high. It is usually manifested in the respect of high TPKI (transactions per kilo instruction). Given a certain read/write ratio, write operations obstruct the LLC port most of the time. This causes the delay of read requests from itself and other concurrent processes.
- (2) **High miss rate:** The cache miss rate is high. The consequence is that most of the data written into the LLC will be useless. Thus, both the time and the energy spent on those LLC writes are wasted.

Fig. 4. The performance speedup after adopting SOAP and changing $MissR_{th}$ from 0.05 to 0.95.

If one process has these two characteristics, it means that this process introduces heavy writes to the LLC but most of them are unnecessary. This type of process obstructs the LLC and degrades the system performance.

Thus, we can set two static thresholds, $Util_{th}$ and $MissR_{th}$; monitor the cache access statistics of each process; and determine which one is likely to be LLC obstructive. First, we can define the utilization of each process according to Equation (1):

$$Utilization = \frac{N_{RD} \times T_{Rd} + N_{WR} \times T_{Wr}}{Execution\ Time}, \quad (1)$$

where N_{RD} and N_{WR} are the read and the write counts, respectively, and T_{Rd} and T_{Wr} represent the latencies of the LLC read and the write operation, respectively. If the calculated value is greater than the threshold $Util_{th}$, this workload is labeled as *high utilization*.

Similarly, we can also monitor the cache miss rate of a process. If it is greater than $MissR_{th}$, we then label that process as *high miss rate*. If one process is both *high utilization* and *high miss rate*, we treat it as a potential *LLC-obstructive* process.

It is relatively simple to implement the SOAP idea, but SOAP has its disadvantages. First, it is highly possible that misjudgment can happen since we only use two simplified metrics. For example, given the criteria used in SOAP, such an “LLC-obstructive” process might not actually block the LLC port for a long time if most of its cache accesses are reads. Though it is possible to overcome this problem by adding more metrics and conditions (e.g., read/write ratio), the scheme itself only becomes marginally better but with more and more hardware complexity.

Second, although the value of $Util_{th}$ and $MissR_{th}$ can be adjusted upfront by the architects, the working thresholds for different processes vary from one case to another, and it becomes very impractical to predetermine a set of universal values. To show the impact of selecting different threshold values, we show a sensitivity study on these two parameters. The details of simulation methodology are described in Section 4. Figure 4 shows how the performance speedup of SOAP² varies when we change the value of $MissR_{th}$ from 0.05 to 0.95 only. This result shows that the speedup can be small if this threshold is set too low. It is because some normal processes would be incorrectly labeled as LLC obstructive when $MissR_{th}$ is low. Their performance drops since they are blocked by the cache management policy and cannot benefit from the existence of the LLC.

On the contrary, Figure 5 shows the speedup of SOAP when $Util_{th}$ is changed from 10% to 90%. This result shows that the speedup can be small if $Util_{th}$ is either too low or too high. If $Util_{th}$ is too low, normal processes can be incorrectly treated as LLC obstructive; vice versa, if $Util_{th}$ is too high, LLC-obstructive processes cannot be

²Each speedup value is the average result among all the benchmarks.

Fig. 5. The performance speedup after adopting SOAP and changing the value of $Util_{th}$ from 10% to 90%.

effectively captured. We see a suboptimal system performance in both of these two cases.

In short, the performance speedup of SOAP is very sensitive to the setting of these thresholds (i.e., $Util_{th}$ and $MissR_{th}$), and it is difficult to predefine the optimal values. Therefore, we stop discussing the details of SOAP any further; instead, we focus the rest of this article on an improved solution called the dynamic obstruction-aware management policy (OAP), which does not require any predefined thresholds.

3.2. A More Practical Approach: Using Heuristic Metric

To make OAP adaptive to different workloads, a more practical way is to use heuristic metrics. The rationale behind adding another level of cache (e.g., L3) into the memory hierarchy is to provide fast-access memory resources so that workloads with good spatial and temporal locality can leverage these memory resources and avoid the time-consuming access to the off-chip DRAM main memory. However, not all the workloads can benefit from this feature.

Assuming that the behavior of a process does not change within a short period of time, if we write data of this process into the LLC, the expected execution time for one of following accesses can be estimated as Equation (2):

$$T = P_{Rd} \times T_{Rd} + (1 - P_{Rd}) \times T_{Wr} + P_{Miss} \times (T_{Mem} + T_{Wr}), \quad (2)$$

where P_{Rd} is the LLC read/write ratio, and P_{Miss} is the LLC miss rate. In addition, T_{Rd} , T_{Wr} , and T_{Mem} are the latency of LLC read, LLC write, and the average latency of main memory, respectively. Note that we assume that this cache uses *fetch-on-write* policy, a widely used policy in modern cache designs.

For each cache miss, new data need to be fetched from the main memory and be written to the LLC at first. Thus, the additional latency is added to the total delay T .

On the other hand, if the data of this process are not written into the LLC, then the following operations have to access main memory directly. Thus, the expected execution time of one access can be estimated as

$$T' = T_{Mem}. \quad (3)$$

If writing the data from one process into the LLC cannot get any benefit in terms of system performance, which means $T > T'$, then we can get

$$P_{Miss} > \frac{T_{Mem} - P_{Rd} \times T_{Rd} - (1 - P_{Rd}) \times T_{Wr}}{T_{Mem} + T_{Wr}}. \quad (4)$$

When the process characteristic satisfies Equation (4) during a period of time, it might cause intensive write operations to the LLC so that writing its data into the LLC will only extend the execution time and hence degrade the system performance. We then can define this type of process as an *LLC-obstructive process*.

We can also observe from Equation (4) that the possibility of LLC obstruction is higher when an STT-RAM LLC is used because the miss rate threshold becomes smaller when

Fig. 6. A three-level cache hierarchy enhanced by obstruction-aware cache management policy (OAP). Newly added structures are highlighted in gray.

the LLC write latency is longer. This is also the reason that the performance degradation is more severe when we use STT-RAM LLCs instead of their SRAM counterparts.

3.3. LLC Obstruction Monitor

To detect LLC-obstructive processes during runtime, we design an obstruction-aware monitor (OAM) and place it before the LLC (i.e., between the L2 and L3 caches). The OAM circuit is separated from the cache hierarchy. This allows the OAM to independently obtain the L3 cache access information from each core.

An adjustable parameter *period* is available in OAM, and it presents the timing granularity of the LLC-obstructive detection. Every *period* is further divided to two parts:

- (1) *sampPeriod*, in which the cache works under the normal policy, and OAMs collect cache access statistics;
- (2) *exePeriod*, in which L3 is managed by OAP using the detection result collected during the last sample period.

In *sampPeriod*, all processes are labeled as *Non-LLC-obstructive*. A per-core OAM collects the statistics including the execution time *currentTime*, the number of read accesses *RD*, the number of write accesses *WR*, and the number of cache misses *Miss*.

At the end of *sampPeriod*, OAM evaluates two metrics: the actual miss rate *MissR* and the obstruction threshold OAP_{th} :

$$MissR = Miss / (RD + WR) \quad (5)$$

$$OAP_{th} = \frac{T_{Mem} - (RD \cdot T_{Rd} + WR \cdot T_{Wr}) / (RD + WR)}{T_{Mem} + T_{Wr}}. \quad (6)$$

According to Equation (4), if *MissR* is larger than OAP_{th} , it means that the process is not benefiting from LLC. Therefore, OAM labels this process as an *LLC-obstructive* process. Otherwise, OAM labels it as a *Non-LLC-obstructive* one. During the following *exePeriod*, depending on whether OAM considers this process LLC obstructive or not, OAP handles the cache accesses from this process accordingly until the next *period* starts. The monitor algorithm can be described in Algorithm 1.

3.4. OAP Architecture

Figure 6 shows a four-core system with a three-level cache hierarchy enhanced by OAP. Similar to previous works [Qureshi and Patt 2006; Xie and Loh 2009], we assume processes are bound to cores. Each core has its own OAM to track L3 cache accesses requested by its private L2 cache and to find out the LLC-obstructive processes. After the potential LLC-obstructive processes are detected, an OAP controller in the shared

Fig. 7. The control flow of an OAP controller. The “LLC obstruction” detection is made by the OAM associated with each core.

ALGORITHM 1: The algorithm of runtime OAM

Input: The new *access* sent to LLC.

Output: The detection result.

Parameters: *period*, *sampPeriod*, *Util_{th}*, *MissR_{th}*.

```

1: if currentTime − startTime == period then
2:   reset all parameters
3:   label as Non-LLC-obstructive
4:   startTime ← currentTime
5: end if
6: if currentTime − startTime < sampPeriod then
7:   if accessIsRead then
8:     RD++ // Update read access count
9:   else
10:    WR++ // Update write access count
11:  end if
12:  if accessIsMiss then
13:    Miss++ // Update miss count
14:  end if
15: end if
16: if currentTime − startTime == sampPeriod then
17:   Calculate OAPth, MissR
18:   if MissR > OAPth then
19:    label as LLC-obstructive
20:  else
21:    label as Non-LLC-obstructive
22:  end if
23: end if
  
```

L3 cache is responsible to prevent LLC-obstructive processes from accessing the L3 cache and ensure data coherence. The control flow of the OAP controller is shown in Figure 7, and its functionality is described as follows:

- L3 read hits*: The L3 cache returns the data to the corresponding L2 cache.
- L3 read misses*: The L3 cache asks for the data from the main memory at first. When the main memory returns the data, the OAP controller checks whether the read requester is LLC obstructive. If it is, the returning data bypasses the L3 and is directly forwarded to the L2. Otherwise, the data is written into the L3 as normal.
- L3 write hits*: The OAP controller checks if the write requester is LLC obstructive at first. If it is, the L3 invalidates the hit data block, and the write request bypasses

Table III. Simulation Settings

Core		4-core, 1.5GHz out-of-order ARM cores
SRAM I-L1/D-L1 caches		private, 32KB/32KB, 16-way, LRU, 64B cache line, write-back, write allocate 2-cycle read, 2-cycle write
SRAM L2 cache		private, 256kB, 8-way, LRU, 64B cache line, write-back, write allocate 8-cycle read, 8-cycle write
L3 cache	Common	shared, 8MB, 8-way, LRU, 4 banks, 8-entry write buffer per bank, 64B cache line, write-back, write allocate
	OAP STT-RAM	17-cycle read, 34-cycle write, w/ OAP
	Baseline STT-RAM	17-cycle read, 34-cycle write, w/o OAP
	Baseline SRAM	17-cycle read, 17-cycle write
DRAM main memory		4GB, 128-entry write buffer, 200-cycle

the L3 and is directly forwarded to the main memory. Otherwise, the data is written into the L3 as normal.

—*L3 write misses*: The OAP controller checks if the requester is LLC obstructive at first. If it is, the write request bypasses the L3 and is forwarded to the main memory directly. Otherwise, the cache line is fetched and allocated at first and then the new data is written into the L3 cache.

Note that we assume the modification is applied on top of a noninclusive/exclusive cache management workflow. If we need to implement OAP atop an inclusive LLC, we should include other techniques, such as adding a bypass buffer [Gupta et al. 2013] to implement cache bypassing.

4. EXPERIMENTAL RESULTS

In this section, we describe our experiment methodology, and we evaluate the performance improvement and the energy reduction achieved by OAP.

4.1. Experiment Methodology

We model a 1.5GHz four-core out-of-order ARMv7 microprocessor using a modified version of gem5 [Binkert et al. 2011]. Our modification to gem5 includes an asymmetric cache read/write latency model, a banked cache model, and a sophisticated cache write buffer scheme. Write buffers are commonly used in conventional cache architecture to hide the performance penalty due to write operations, but the write buffer size cannot be too large because of its fully associative look-up overhead [Sun et al. 2009]. Thus, the write buffer technique alone is not sufficient to deal with the performance degradation due to the STT-RAM long write latency. We use an eight-entry write buffer in this work.

Simulation setting details are listed in Table III. All the circuit-level cache module parameters (e.g., read latency and write latency) are obtained from NVSim [Dong et al. 2012] and are consistent with Table I. The simulation workloads are from the SPEC CPU2006 benchmark suite [SPEC CPU 2006]. Workloads are compiled using gcc version 4.5.2 (Sourcery G++ Lite 2011.03-41) without any optimization setting. We simulate each experiment for at least 2 billion instructions. We set the OAM sampling period to be 0.1 million cycles and OAM launching period to be 10 million cycles.

We use IPC (instructions per cycle) as the performance metric and use geometric mean to average the performance of cores and derive the speedup metric

Fig. 8. The performance speedup over conventional STT-RAM L3-based system after adopting OAP with duplicated and mixed workloads in a four-core system, where *benchmark-dup4* means executing *benchmark* on each core.

Table IV. The Workload List in the Mixed Groups

Mixed group	Workloads
mix1	lbm+libquantum+sjeng+cactusADM
mix2	povray+mcf+namd+lbm
mix3	bwaves+cactusADM+astar+hmmer
mix4	gcc+lbm+libquantum+bwaves
mix5	cactusADM+gcc+mcf+lbm
mix6	libquantum+bzip2+bwaves+cactusADM
mix7	lbm+sjeng+bzip2+libquantum
mix8	mcf+bwaves+sjeng+lbm

($speedup = geomean(\frac{IPC_i}{IPC_{baseline}})$), which accounts for both fairness and performance [Xie and Loh 2009].

4.2. Performance Speedup

Figure 8 illustrates the speedup over a conventional STT-RAM L3-based system after adopting the OAP architecture. The first eight groups run four duplicated processes, and the other eight groups are workload mixtures as listed in Table IV. It shows that after adopting OAP on the STT-RAM L3 cache, the system performance is improved by 14% on average (up to 42%) compared to the conventional cache management policy.

Furthermore, we list several characteristics over workloads in Table V: (1) the percentage of reads and writes to L3 cache, (2) the miss rate of L3, and (3) the port utilization calculated as Eq. (1). In order to understand where the performance saving comes from, the frequency analysis for the different paths outlined in the control flow of the OAP controller can be calculated based on the data in Table V. Generally, the workloads with a high write miss rate and high port utilization are more likely to be detected as LLC-obstructive processes, and the groups that have more LLC-obstructive processes in more periods benefit more from OAP architecture. Because the OAM can quickly detect such processes during sampling periods, the OAP controller skips their unnecessary writes to the STT-RAM L3 cache.

For groups with a mixed workload, the performance improvement comes from two respects. First, the performance of LLC-obstructive processes is improved because OAP skips the unnecessary writes in these processes and mitigates the penalty coming from the longer write latency. Second, the performance of concurrent processes is also improved since the obstruction on the L3 port is reduced and their requests to L3 can be satisfied more quickly. In addition, the available cache lines of L3 are

Table V. The Characteristics of Each Workload

Workloads	L3 read	L3 write	L3 miss rate	L3 utilization
bwaves-dup4	32%	68%	42%	0.49
bzip2-dup4	39%	61%	36%	0.30
cactusADM-dup4	42%	58%	95%	0.44
libquantum-dup4	60%	40%	99%	0.48
gcc-dup4	74%	26%	73%	0.09
lbm-dup4	23%	77%	89%	0.63
mcf-dup4	72%	28%	74%	0.34
sjeng-dup4	46%	54%	94%	0.16
mix1	43%	57%	96%	0.56
mix2	44%	56%	69%	0.28
mix3	36%	64%	73%	0.37
mix4	42%	58%	78%	0.57
mix5	45%	55%	80%	0.46
mix6	42%	58%	73%	0.51
mix7	43%	57%	83%	0.47
mix8	40%	60%	76%	0.45

Fig. 9. The IPC of systems with SRAM L3, conventional STT-RAM L3, and OAP STT-RAM L3 (normalized to the value of SRAM L3-based system).

increased for these well-behaved processes because the number of write operations from LLC-obstructive processes is reduced, and thus it increases their hit rate and performance.

4.3. Compare to SRAM LLC

Adopting OAP architecture in L3 cache makes STT-RAM more competitive compared to SRAM. To evaluate its benefits, we simulate another baseline system with a shared SRAM L3 cache, which is configured as Table III.

Figure 9 shows the normalized IPC of the systems with normal SRAM L3, STT-RAM L3, and OAP STT-RAM L3. The result shows that compared to the SRAM L3-based system, a straightforward STT-RAM L3 replacement degrades the system performance by 13.2% on average due to its longer write latency. However, after adopting OAP, the performance of the STT-RAM L3-based system can be improved significantly. Compared to SRAM L3, the performance degradation in the OAP STT-RAM-based system is only 0.7% on average.

Besides the performance improvement achieved by OAP, using STT-RAM L3 itself can save energy as well. This is because the leakage power is dominant in the L3 cache and the leakage power of STT-RAM is only about 1% of SRAM's. In addition, adopting OAP can reduce the total energy consumption further because it reduces the number of writes, which is a major source of the dynamic energy in STT-RAM caches. Figure 10 shows the normalized energy consumption of three different systems, and

Fig. 10. The energy consumption of SRAM L3, conventional STT-RAM L3, and OAP STT-RAM L3 (normalized to the value of SRAM L3). Each value is broken down to leakage energy and dynamic energy.

each value is broken down to leakage energy and dynamic energy. Due to the leakage energy reduction, an 8MB STT-RAM L3 can save around 90% total energy compared to an SRAM L3 cache with the same capacity. And after adopting OAP architecture, the energy of STT-RAM L3 is further reduced by 64% on average.

The final advantage of STT-RAM L3 cache is the silicon area reduction. It is not related to OAP but is also important. Compared to an 8MB SRAM cache, an 8MB STT-RAM cache can save 72.5% of the area as shown in Table I.

In summary, STT-RAM L3 enhanced by OAP makes itself a more attractive option in replacing SRAM L3 cache.

5. OVERHEAD ANALYSIS

5.1. Hardware Overhead

We evaluate the hardware overhead of the proposed OAP architecture using the Synopsys Design Compiler with a 32nm CMOS library. According to the synthesis result, the OAM circuitry only incurs a 0.05mm^2 area overhead, which is negligible compared to the L3 cache area (usually half of the total chip area). According to the energy analysis, the extra energy consumption per access is about 1.7pJ , which is included in our simulations. In addition, it is straightforward to integrate the OAP judgment flow into the conventional cache controller using bypass circuits.

In terms of latency overhead, considering that OAMs are separated from the cache hierarchy and the OAP controller only brings some branch decisions in the control flow, we do not add any latency overhead in our simulation settings.

5.2. Impact on L3 Miss Rate

OAP avoids the writes to STT-RAM L3 from LLC-obstructive workloads, but it might increase the miss rate since the data of LLC-obstructive processes are not allocated in L3. If the L3 miss rate increases significantly, the system performance might be degraded because more accesses to the main memory are needed. However, the OAP scheme is designed to detect LLC-obstructive processes automatically, which usually have a high miss rate. Figure 11 shows the normalized L3 miss rate of an OAP system compared to the baseline. It shows that the overall miss rate only increases by 6%. Thus, the OAP scheme has a small impact on system miss rates.

5.3. Traffic Overhead Analysis

We also need to evaluate the traffic overhead since some writes are bypassed from LLC in the OAP system. Figure 12 and Figure 13 show the main memory traffic and the L3 traffic of the OAP system normalized to the baseline system, respectively.

Fig. 11. The L3 miss rate comparison between a baseline system and an OAP system.

Fig. 12. Read and write access numbers of the main memory normalized to the baseline system.

Fig. 13. Read and write access number of L3 cache normalized to baseline system.

From Figure 12, we can observe that OAP increases the overall read and write accesses to main memory by 39% and 10%, respectively. Although the traffic to the main memory is increased, the number of write accesses to L3 is reduced by 62% on average as shown in Figure 13. Therefore, with these two factors combined, the performance of the entire system is increased after adopting OAP since the port obstruction possibility becomes lower, and the write latency of the STT-RAM L3 cache is long.

6. SENSITIVITY STUDY

In this section, we describe our experiments and results to study the performance sensitivity in terms of different write latencies, bank numbers, core numbers, and the period length of OAM when applying the OAP technique.

Fig. 14. The performance speedup over conventional STT-RAM L3-based system after adopting OAP architecture with duplicate and mixed workloads in four-core systems with different cache write latency, in which *benchmark-dup4* means executing *benchmark* on each core.

6.1. Sensitivity to Cache Write Latency

In Section 4, we use the cache configuration shown in Table III, in which the LLC latency is consistent with Table I. We expect that OAP also works effectively on caches with different cache configurations and can get more benefits when increasing the STT-RAM LLC latency.

We run experiments on different LLC write latencies ranging from 15ns to 30ns. Figure 14 shows the results. Among all the workloads, the performance speedup is 1.06 on average and up to 1.21 when the LLC write latency is reduced to 15ns. On the other hand, the effectiveness of OAP is more significant when we increase the LLC latency. The system speedup is increased to 1.24 and 1.33 on average when the LLC write latency is increased to 25ns and 30ns, respectively.

Generally, adopting the OAP technique gets more performance benefit when the STT-RAM LLC latency is longer. The reason is that the port obstruction problem is more severe when L3 write latency is longer. An ongoing write from one core could have a higher chance to block other reads to the same cache bank. The OAP technique improves the system performance by detecting LLC-obstructive processes and skipping their unnecessary writes to the STT-RAM L3 cache.

6.2. Sensitivity to Bank Number

Large on-chip caches are usually split to multiple banks to increase access parallelism. If the LLC has more banks, the possibility of port obstruction will be reduced since an ongoing write operation only blocks the port of the corresponding cache bank and does not affect the port of other cache banks. To study the effect of the LLC bank number, we adopt OAP in systems with different L3 configurations as one bank, two banks, four banks, and eight banks, respectively. All the other system parameters are consistent with the ones in Table III.

Figure 15 shows the performance speedup result. Among all workloads, the speedup is as high as 2.08 and 1.46 on average when the L3 cache is configured as a one-bank and two-bank system, respectively. The average speedup is 1.02 when the bank number is increased to eight.

Therefore, the system with a smaller LLC bank number can get more performance improvement by adopting the OAP technique. The reason is that a multibank organization provides more parallelism and mitigates the port obstruction problem. However, adding bank numbers is expensive in terms of cost. Thus, adopting OAP to improve the performance for the system with a smaller bank number is a better solution.

Fig. 15. The performance speedup over conventional STT-RAM L3-based system after adopting OAP architecture with duplicate and mixed workloads in four-core systems with different L3 bank numbers, in which *benchmark-dup4* means executing *benchmark* on each core.

Fig. 16. The performance speedup over conventional STT-RAM L3-based system after adopting OAP architecture with duplicate and mixed workloads in systems with different core number, in which *benchmark-dup* means executing *benchmark* on each core.

6.3. Sensitivity to Core Number

Another sensitivity study is targeted to the core number. The simulation result given in Section 4 is focused on a four-core system, but OAP can also be adopted in single-core or multicore systems with different core numbers.

We run simulations on systems with one core, two cores, and eight cores. Other system configurations are not changed and are shown in Table III. For the single-core system, we only execute one program without using the mixed groups. For the two-core system, we use the first two programs in each mixed group as shown in Table IV. For the eight-core system, we duplicate each program to two cores for each mixed group.

Figure 16 shows the speedup results. The speedup of the single core system is 1.01 on average and up to 1.07. This speedup is not as significant as the four-core system because there is no negative effect between different applications, and the obstruction problem is not as severe as in multicore systems.

For the two-core and eight-core systems, the overall speedup is 1.06 and 1.32, respectively. It shows that when the core number is increased, the problem of LLC port obstruction is more and more severe since there are more processes sharing the same port of LLC. Thus, adopting the OAP technique can get more performance improvement with the core number increasing.

6.4. Sensitivity to the Length of Launching Period and Sampling Period

In addition, we run a sensitivity study on the length of the launching period and sampling period. If the sampling period is too long compared to the launching period, the running time using the OAP technique is short and the effectiveness of OAP is

Fig. 17. The performance speedup over conventional STT-RAM L3-based system after adopting OAP architecture with duplicate and mixed workloads in systems with different lengths of launching period and sampling period, in which *benchmark-dup* means executing *benchmark* on each core.

reduced. On the other hand, if the sampling period is too short, the LLC-obstructive processes might not be captured by OAM and the performance speedup is decreased.

Figure 17 shows the performance speedup of OAP architecture with different lengths of launching period and sampling period. It shows that the overall performance speedup is reduced to 13% when the sampling period is too small (i.e., the launching period is set as 10M cycle and the sampling period is set as 0.01M cycle).

7. RELATED WORK

Some previous work focused on how to mitigate the performance penalty incurred by asymmetrically long write latencies of STT-RAM. Xu et al. [2011] proposed a dual write speed scheme to improve the average access time of the STT-RAM cache. An early write termination scheme was proposed to reduce the unnecessary writes to STT-RAM cells [Zhou et al. 2009]. Sun et al. [2009] proposed a hybrid cache architecture with read-preemptive write buffers. Smullen et al. [2011] and Sun et al. [2011] traded off the STT-RAM nonvolatility to improve the write speed and the write energy. These works are all focused on modifying STT-RAM cell or cache architecture designs but not the cache management policy. Therefore, they are compatible with our work and it is possible to use them together to improve the system performance further.

There are also some previous works focused on cache management policies to improve the overall performance. Some tried to find the optimal partition and maximize system performance [Qureshi and Patt 2006; Kim et al. 2004; Srikantaiah et al. 2009]. Xie and Loh [2009] proposed a pseudo-partitioning which combines targeted insertion and incremental promotion without enforcing the target partition. Jaleel et al. [2010] proposed a technique to identify the access pattern and prevent blocks with a distant reference interval from polluting the cache. However, these works focused on SRAM caches, where there is no read/write asymmetric problem, and they always allocate more than one cache way for each process. In this work, we tackled these two problems together and focused on how to improve the basic cache management policy for STT-RAM last-level caches.

8. CONCLUSION

While the scaling of SRAM and eDRAM is constrained by cell density and leakage energy, the emerging STT-RAM memory technology is explored as one of the potential alternatives of last-level caches. Despite STT-RAM having higher density and smaller leakage energy, STT-RAM has a much longer write latency compared to SRAM and may cause severe performance degradation in a multicore system where cache accesses are obstructed by the write operations. In this article, we proposed OAP, an

obstruction-aware cache management policy, which can significantly improve system performance and reduce energy consumption across different workloads for future STT-RAM last-level cache design.

REFERENCES

- Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *SIGARCH Computer Architecture News* 39, 2, 1–7.
- Xiangyu Dong, Wu Xiaoxia, Sun Guangyu, Xie Yuan, Li Helen, and Chen Yiran. 2008. Circuit and microarchitecture evaluation of 3D Stacking Magnetic RAM (MRAM) as a Universal Memory Replacement. In *Design Automation Conference*. 554–559.
- Xiangyu Dong, Cong Xu, Yuan Xie, and Norm Jouppi. 2012. NVSim: A circuit-level performance, energy, and area model for emerging non-volatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 0 (2012).
- A. Driskill-Smith. 2011. Latest advances and future prospects of STT-RAM. In *NVMW*.
- S. Gupta, Hongliang Gao, and Huiyang Zhou. 2013. Adaptive cache bypassing for inclusive last level caches. In *Proceedings of the 27th International Symposium on Parallel Distributed Processing (IPDPS)*. 1243–1253.
- Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, Jr., and Joel Emer. 2010. High performance cache replacement using re-reference interval prediction (RRIP). In *Proceedings of the International Symposium on Computer Architecture*. 60–71.
- T. Kawahara and others. 2007. 2Mb spin-transfer torque RAM (SPRAM) with bit-by-bit bidirectional current write and parallelizing-direction current read. In *ISSCC*.
- Chris H. Kim, Jae-Joon Kim, Saibal Mukhopadhyay, and Kaushik Roy. 2003. A forward body-biased low-leakage SRAM cache: device and architecture considerations. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 6–9.
- Seongbeom Kim, Dhruva Chandra, and Yan Solihin. 2004. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*. 111–122.
- David Kroft. 1981. Lockup-free instruction fetch/prefetch cache organization. In *Proceedings of the International Symposium on Computer Architecture*. 81–87.
- T. M. Maffitt, J. K. DeBrosse, J. A. Gabric, E. T. Gow, M. C. Lamorey, J. S. Parenteau, D. R. Willmott, M. A. Wood, and W. J. Gallagher. 2006. Design considerations for MRAM. *IBM Journal of Research and Development* 50, 1 (2006), 25–39.
- Kumiko Nomura, Keiko Abe, Hiroaki Yoda, and Shinobu Fujita. 2011. Ultra low power processor using perpendicular-STTMRAM/SRAM based hybrid cache toward next generation normally-off computers. In *MMM*.
- Moinuddin K. Qureshi and Yale N. Patt. 2006. Utility-Based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the International Symposium on Microarchitecture*. 423–432.
- Hari M. Rao and Jung Pill Kim. Multi-port non-volatile memory that includes a resistive memory element. U.S. Patent Application 13/772,411.
- Clinton W. Smullen, Mohan Vidyabhushan, Anurag Nigam, Sudhanva Gurumurthi, and Mircea R. Stan. Relaxing non-volatility for fast and energy-efficient STT-RAM caches. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA)*. 50–61.
- SPEC CPU. 2006. SPEC CPU2006. Retrieved from <http://www.spec.org/cpu2006/>.
- S. Srikantaiah, M. Kandemir, and Qian Wang. 2009. Sharp control: Controlled shared cache management in chip multiprocessors. In *Proceedings of the International Symposium on Microarchitecture*. 517–528.
- Sun Guangyu, Xiangyu Dong, Yuan Xie, Jian Li, and Yiran Chen. 2009. A novel architecture of the 3D stacked MRAM L2 cache for CMPs. In *High Performance Computer Architecture*. 239–249.
- Sun Zhenyu, Xiuyuan Bi, Hai Helen Li, Weng-Fai Wong, Zhong-Liang Ong, Xiaochun Zhu, and Wenqing Wu. 2011. Multi retention level STT-RAM cache designs with a dynamic refresh scheme. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. 329–338.
- Thoziyoor Shyamkumar, Jung Ho Ahn, Matteo Monchiero, Jay B. Brockman, and Norman P. Jouppi. 2008. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In *35th International Symposium on Computer Architecture (ISCA'08)*. 51–62.

- Kenji Tsuchida, Tsuneo Inaba, Katsuyuki Fujita, Yoshihiro Ueda, Takafumi Shimizu, Yoshiaki Asao, Takeshi Kajiyama, Masayoshi Iwayama, Kuniaki Sugiura, Sumio Ikegawa, Tatsuya Kishi, Tadashi Kai, Minoru Amano, Naoharu Shimomura, Hiroaki Yoda, and Yohji Watanabe. 2010. A 64Mb MRAM with clamped-reference and adequate-reference schemes. 2010. In *International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 258–259.
- Yuejian Xie and Gabriel H. Loh. 2009. PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches. In *Proceedings of the International Symposium on Computer Architecture*. 174–183.
- Xu Wei, Hongbin Sun, Xiaobin Wang, Yiran Chen, and Tong Zhang. 2011. Design of last-level on-chip cache using spin-torque transfer RAM (STT RAM). *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19, 3, 483–493.
- Ping Zhou, Zhao Bo, Jun Yang, and Youtao Zhang. 2009. Energy reduction for STT-RAM using early write termination. In *International Conference on Computer-Aided Design-Digest of Technical Papers, 2009*. 264–268.

Received July 2013; revised April 2014; accepted April 2014