OAP: An Obstruction-Aware Cache Management Policy for STT-RAM Last-Level Caches

Jue Wang *, Xiangyu Dong †, Yuan Xie*‡
* Pennsylvania State University, USA
† Qualcomm Technology, Inc., USA
‡ AMD Research, Advanced Micro Devices, Inc., USA
Email: *{jzw175, yuanxie}@cse.psu.edu, †xiangyud@qti.qualcomm.com,‡yuan.xie@amd.com

Abstract—**Emerging memory technologies are explored as potential alternatives to traditional SRAM/DRAM-based memory architecture in future microprocessor designs. Among various emerging memory technologies, Spin-Torque Transfer RAM (STT-RAM) has the benefits of fast read latency, low leakage power, and high density, and therefore has been investigated as a promising candidate for last-level cache (LLC)[1]. One of the major disadvantages for STT-RAM is the latency and energy overhead associated with the write operations. In particular, a long-latency write operation to STT-RAM cache may obstruct other cache accesses and result in severe performance degradation. Consequently, mitigation techniques to minimize the write overhead are required in order to successfully adopt this new technology for cache design. In this paper, we propose an obstruction-aware cache management policy called OAP. OAP monitors the cache to periodically detect LLC-obstruction processes, and manage the cache accesses from different processes. The experimental results on a 4-core architecture with an 8MB STT-RAM L3 cache shows that the performance can be improved by 14% on average and up to 42%, with a reduction of energy consumption by 64%[2].**

I. Introduction

While computational components (cores) have continuously benefited from the scaling offered by Moore's law, it is widely recognized that the memory system has been a laggard/bottleneck in terms of both performance and power. The integration of multiple cores on a single die accentuates the already daunting *memory-wall* problem. Consequently, the memory hierarchy design has to play 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. As a result, deeper cache hierarchy and larger cache capacity has been observed in contemporary microprocessor designs. For example, Intel i7-3930K processor is equipped with a 12MB SRAM L3 cache, and IBM POWER7 processor has an unprecedented 32MB embedded DRAM (eDRAM) L3 cache. However, conventional SRAM or eDRAM are facing constraints of cell area and leakage energy consumption with technology scaling.

Recently, new memory technologies such as *Spin-Torque Transfer RAM (STT-RAM), Phase-change RAM(PCRAM), and Resistive RAM (ReRAM)* are being explored as potential alternatives of SRAM and DRAM. While PCRAM and ReRAM have the wear-out issues and therefore are not suitable to be used as on-chip cache, it is attractive to use STT-RAM as last-level cache (LLC) because of its fast read latency, high density, low leakage, and non-volatility for energy savings. Toshiba already revealed their plan in using STT-RAM to replace a 512KB SRAM L2 cache for low power purpose [1]. Another example is that replacing DRAM with STT-RAM in data centers can reduce power by up to 75% [2].

Unfortunately, while incorporating STT-RAM cache has many advantages, one of the major disadvantages for STT-RAM is the latency and energy overhead associated with the write operations. In particular, a long-latency write operation to STT-RAM cache may obstruct other cache accesses [3], [4], especially when multiple processes are running in a multi-core systems, and result in severe performance degradation.

Consequently, mitigation techniques to minimize the write overhead are critical in order to successfully adopt this new technology for cache design. In this paper, we propose a cache management policy for STT-RAM last-level cache (LLC). The key contributions can be summarized as follows:

- We define a type of workload characteristics called LLC-obstruction. Workloads with this characteristic have significant performance degradation in STT-RAM LLC based system, and they also negatively affect the performance of other processes.
- We design an obstruction-aware monitoring mechanism, OAM, which detects the behavior of LLC-obstruction processes during runtime.
- We propose an obstruction-aware cache management, OAP, which significantly improves performance and reduces energy consumption across different workloads with negligible hardware overhead.

The proposed OAP approach significantly improves the performance and reduces energy consumption for future multi-core systems using STT-RAM last-level cache design. Our experiment results show that the performance of a projected 4-core system with an 8MB STT-RAM LLC cache is improved by 14% on average and up to 42% after adopting OAP, and the energy consumption can be reduced by 64%. Consequently, our proposed method can effectively mitigate the overhead associated with the write operations in STT-RAM cache, and pave the way for the adoption of using STT-RAM for future microprocessor design.

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

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

978-3-9815370-0-0/DATE13/© 2013 EDAA

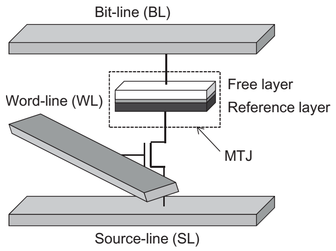

Fig. 1. The conceptual view of an STT-RAM cell.

II. Background and Motivation

A. STT-RAM Technology

Spin-Torque Transfer RAM (STT-RAM) is a new type of Magnetic RAM (MRAM). The basic storage element in STT-RAM is a magnetic tunnel junction (MTJ), in which a thin tunneling dielectric, is sandwiched by two ferromagnetic layers, as shown in Figure 1. One ferromagnetic layer ("reference layer") is designed to have a fixed magnetization direction, while the magnetization of the other layer ("free layer") can be flipped by a switching current. An MTJ has a low (high) resistance if the magnetization direction of the free layer and the reference layer are parallel (anti-parallel), so that a storage of "0" or "1" can be represented.

B. Using STT-RAM for Last-Level Caches

Compared to the conventional SRAM-based cache design, the advantages of the STT-RAM cache are smaller area and lower leakage power. The cell size of STT-RAM is currently in the range of $13F^2$ [5] to $100F^2$ [6] (where F is the feature size), which is much smaller than the SRAM size (e.g. $146F^2$ [7]) and can greatly improve the cache capacity. In addition, STT-RAM has zero leakage power consumption from memory cells due to its non-volatility. Previous work [8] shows the leakage energy can be as high as 80% of total energy consumption for an L2 cache in 130nm process. Thus, using STT-RAM last-level caches can effectively reduce standby leakage power in caches.

Table I shows the characteristics of a 4-bank 8-way 8MB SRAM cache and its STT-RAM counterpart. The estimation is given by NVSim [9], a performance, energy, and area model for emerging non-volatile memory technologies. It shows that STT-RAM cache has much smaller area and leakage power than the one of SRAM. However, as also shown in Table I, the write latency and energy of STT-RAM are much larger than that of SRAM caches. Such overheads are the major obstacle of adopting the emerging STT-RAM technology for cache designs.

C. Motivation

As the last-level cache, L3 is usually implemented using single-port memory bitcell due to the infeasible cost associated with multi-port bitcell designs. For example, building dual-port STT-RAM cell requires at least 4 transistor [10] (a 4X 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 blocking time for the L3 cache port and delay all the following read operations that are on the critical path from the performance aspect. In addition, when an STT-RAM shard L3 is used in a multi-core system, an ongoing write from one core could block all the reads to the same cache bank from other cores, and it greatly reduces the overall system performance.

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

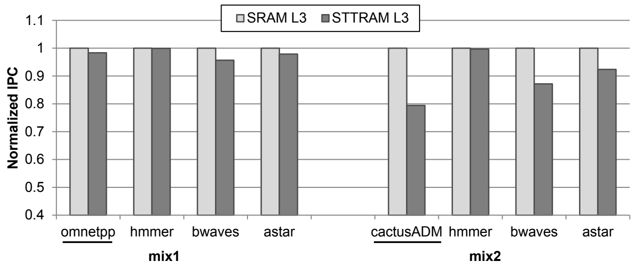

Fig. 2. Normalized IPC of systems with SRAM L3 and STT-RAM L3 for two groups of 4 mixed workloads, in which only the first core runs different workload and the others are the same.

To illustrate this problem, we design two workload mixtures, *mix1* and *mix2* (see Section IV for details on simulation methodology). In *mix1*, we run 4 LLC-friendly workloads (*omnetpp*, *hmmer*, *bwaves*, and *astar*) on the 4-core system; In *mix2*, we only substitute workload *omnetpp* with *cactusADM*. Figure 2 shows the normalized IPC (instructions per cycle) of each process in these two mixtures using SRAM L3 and STT-RAM L3, respectively. As shown, for the *mix1* group, the IPC of each process is reduced less than 5% when replace SRAM L3 with STT-RAM L3. However, for the *mix2* group, not only the core running *cactusADM* has an IPC loss of 21% after using STT-RAM L3, but the IPCs of the other 3 cores are degraded by 8%-13% as well.

Thus, we define a special type of process characteristic called **LLC-obstruction**. Like *cactusADM* in Figure 2, if an LLC-obstruction process running in a multi-core system with STT-RAM L3 cache, it will not only experience a performance loss itself, but also negatively affects the performance of other processes running simultaneously on the system. Therefore, we need a new STT-RAM L3 cache management policy, and its task is to first identify any potential LLC-obstruction processes on-the-fly and then avoid the performance degradation caused by such LLC-obstruction processes.

III. Obstruction-Aware Cache Management Policy

A. Heuristic Metric

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 spacial and temporal locality can leverage these memory resources in L3 and

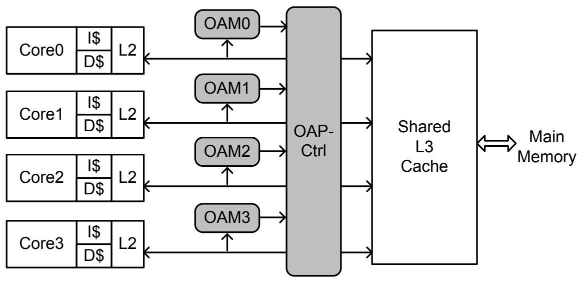

Fig. 3. A 3-level cache hierarchy enhanced by obstruction-aware cache management policy (OAP). Newly added structures are shaded.

avoid the time-consuming access to the off-chip DRAM main memory. However, not all the processes benefit from this feature.

Assuming that the behavior and characteristic of a process are not changed during a short period of running time, if we write data of this process into LLC, the expected execution time for one of following accesses can be estimated as Equation 1, in which, P_{Rd} is the read/write ratio, P_{Miss} is the miss rate of LLC. T_{Rd}, T_{Wr}, T_{Mem} are the latency of LLC read, LLC write, and the average latency of main memory, respectively. Note that we assume the cache uses *fetch-on-write* policy, which is a widely-used policy in modern cache designs.

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

For each access miss, new data need to be fetched from main memory and be written to 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 LLC, then the following operations need to access main memory directly. Thus, the expected execution time for one access can be estimated as:

$$T' = T_{Mem} \tag{2}$$

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

$$P_{Miss} > \frac{T_{Mem} - P_{Rd} \times T_{Rd} - (1 - P_{Rd}) \times T_{Wr}}{T_{Mem} + T_{Wr}} \tag{3}$$

When the process characteristic satisfies Equation 3 during a period of runtime, it might cause intensive write operations to LLC so that writing its data into LLC will extend the execution time and degrade the system performance. We define this type of processes as *LLC-obstruction processes*.

We can also observe from Equation 3 that the possibility of LLC-obstruction is higher when STT-RAM LLC is used, because the miss rate threshold becomes smaller when T_{Wr} is larger. It is also the reason that the performance degradation problem is more serious when we use STT-RAM LLC instead of its SRAM counterpart.

B. LLC Obstruction Monitor

To detect LLC-obstruction processes during runtime, we design the obstruction-aware monitor (OAM) which is placed before LLC (i.e. between the L2 and L3 caches). The OAM circuit is separated from the cache hierarchy, which allows OAM to independently obtain the L3 cache access information from 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-obstruction
 4:     $startTime \leftarrow currentTime$
 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 $OAP_{th}$, $MissR$
18:     if $MissR > OAP_{th}$ then
19:         label as LLC-obstruction
20:     else
21:         label as Non-LLC-obstruction
22:     end if
23: end if
```

A tunable parameter $period$ is set in OAM, and it presents the period length of launching an LLC-obstruction detection. Every $period$ is divided to two parts: $sampPeriod$, in which the cache works under the normal policy and OAMs collect statistics; $exePeriod$, 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-obstruction*. A per-core OAM collects statistic including: 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 calculates two parameters: the actual miss rate *MissR* and the obstruction threshold OAP_{th}.

$$MissR = Miss/(RD + WR) \tag{4}$$

$$OAP_{th} = \frac{T_{Mem} - (RD \cdot T_{Rd} + WR \cdot T_{Wr})/(RD + WR)}{T_{Mem} + T_{Wr}} \tag{5}$$

According to Equation 3, if the result of *MissR* is larger than OAP_{th}, OAM labels this process as *LLC-obstruction*. Otherwise, OAM labels it as *Non-LLC-obstruction*. Then during the following $exePeriod$, this process is treated as the OAM detection result until the next $period$ starts. The monitor algorithm is described in Algorithm 1.

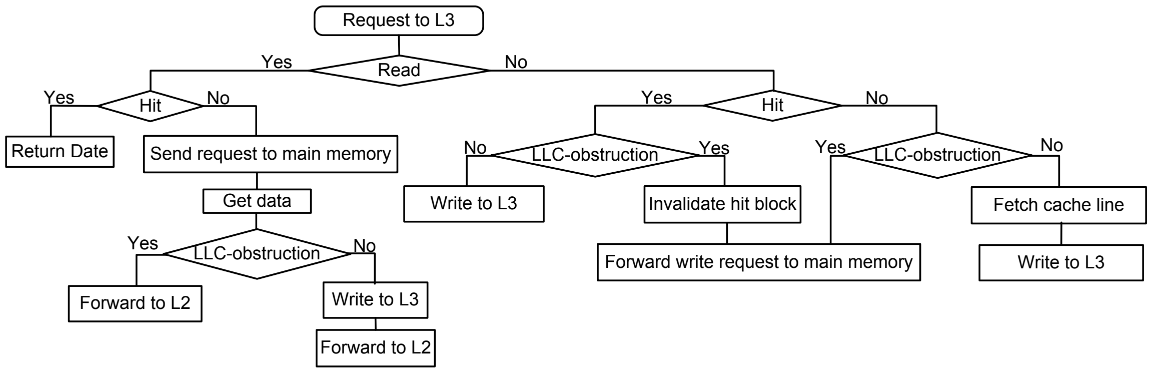

Fig. 4. The control flow of OAP-controller. The detection of "LLC-obstruction" is based on the OAM assocated with each core.

C. Architecture of OAP

Figure 3 shows a 4-core system with a 3-level cache hierarchy enhanced by OAP. Similar to previous works [11], [12], 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 find out the LLC-obstruction processes. After the potential LLC-obstruction processes are detected, an OAP-controller in the shared L3 cache is responsible to prevent LLC-obstruction processes from accessing L3 cache and also make sure data coherence. The control flow of the OAP-controller is shown as Figure 4, and its functionality is described as follows:

- *L3 read hits*: L3 cache returns the data to the corresponding L2 cache.
- *L3 read misses*: L3 cache asks the data from main memory at first. When main memory returns the data, the OAP-controller checks weather the read requester is LLC-obstruction. If it is, the returning data bypasses L3 and is directly forwarded to L2. Otherwise, the data is written into L3 as normal.
- *L3 write hits*: the OAP-controller checks if the write requester is LLC-obstruction at first. If it is, L3 invalidates the hit data block, and the write request bypasses L3 and is directly forwarded to main memory. Otherwise, the data is written into L3 as normal.
- *L3 write misses*: the OAP-controller checks if the requester is LLC-obstruction firstly. If it is, the write request bypasses L3 and is forwarded to main memory directly. Otherwise, the cache line is fetched and allocated at first and then the new data is written into L3 cache.

D. Hardware Overhead

We evaluate the hardware overhead of the proposed OAP architecture using Synopsys Design Compiler with a 32nm CMOS library. According to the synthesis result, the OAM circuitry only incurs $0.05mm^2$ 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 also included in our simulations. In addition, it is straightforward to integrate the OAP judgment flow into the conventional cache controller using some bypass circuits.

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

IV. Experimental Results

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

A. Experiment Methodology

We model a 1.5GHz 4-core out-of-order ARMv7 microprocessor using our modified version of gem5 [13]. Our modification to gem5 includes an asymmetric cache read/write latency model, a banked cache model, and a sophisticated cache write buffer scheme. Write-buffer technique is 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 [4]. Thus, the write buffer technique alone is not sufficient to deal with the performance degradation due to STT-RAM long write latency. We use an 8-entry write buffer in this work.

Simulation setting details are listed in Table II. All the circuit-level cache module parameters (e.g. read latency and write latency) are obtained from NVSim [9] and are consistent with Table I. The simulation workloads are from SPEC CPU2006 benchmark suite [14]. We simulate each experiment for at least 2 billion instructions. The OAM sampling period is set to be 0.1 million cycles and its launching period is set 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 ($speedup = geomean(\frac{IPC_i}{IPC_i^{baseline}})$) which accounts for both fairness and performance [12].

B. Performance Speedup

Figure 5 illustrates the speedup over conventional STT-RAM L3 based system after adopting the OAP architecture. The first 8 groups run 4 duplicated processes, and the other 8 groups are workload mixtures as listed in Table III. It shows that after adopting OAP on the STT-RAM L3 cache, the

TABLE II
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

TABLE III
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

system performance is improved by 14% on average (up to 42%) compared to the conventional cache management policy.

Generally, the groups which have more LLC-obstruction processes in more periods benefit more from OAP architecture. Because OAM can quickly detect such processes during sampling periods, and OAP-controller skips their unnecessary writes to the STT-RAM L3 cache. For groups with mixed workload, the performance improvement comes from two respects. First, the performance of LLC-obstruction 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 is increased for these well-behaviored processes because the number of write operations from LLC-obstruction processes is reduced, thus it increases their hit rate and performance.

C. 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 II.

Figure 6 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 SRAM L3 based system, the conventional STT-RAM L3 degrades the system performance by 13.2% on average due to its longer write latency. However, after adopting the OAP architecture, the performance of STT-RAM L3 based system is improved significantly. Compared

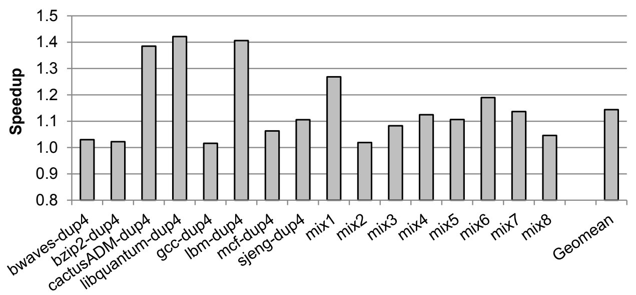

Fig. 5. The performance speedup over conventional STT-RAM L3 based system after adopting OAP architecture with duplicate and mixed workloads in a 4-core system, in which *benchmark-dup4* means executing *benchmark* on each core.

to SRAM L3, the performance degradation in OAP STT-RAM based system is only 0.7% on average.

Besides the performance improvement achieved by OAP, using STT-RAM L3 itself can bring us energy savings. It is because the leakage power is dominant in 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 7 shows the normalized energy consumption of 3 different systems and 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 architecutre, 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 also important. Compared to an 8MB SRAM cache, an 8MB STTRAM cache can save 72.5% 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.

V. 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.* [15] proposed a dual write speed scheme to improve the average access time of STT-RAM cache. Smullen *et al.* [16] and Sun *et al.* [17] traded off the STT-RAM non-volatility to improve the write speed and the write energy. Sun *et al.* [4] proposed a hybrid cache architecture with read-preemptive write buffers. An early write termination scheme was proposed to reduce the unnecessary writes to STT-RAM cells [18]. However, these works are all focused on modifying STT-RAM cell or cache architecture designs but not the cache management policy. There are also some previous work focused on cache management policies to improve the overall performance. Some tried to find the optimal partition and maximize system performance [11], [19], [20]. Xie *et al.* proposed a pseudo-partitioning which combines targeted insertion and incremental promotion without enforcing the target partition [12]. Jaleel *et al.* proposed a

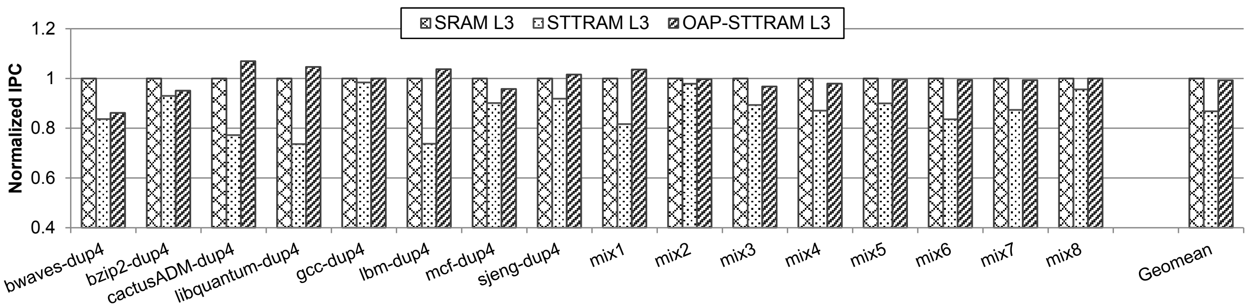

Fig. 6. The IPC of systems with SRAM L3, conventional STT-RAM L3 and OAP STT-RAM L3 (normalized to the value of SRAM L3 based system).

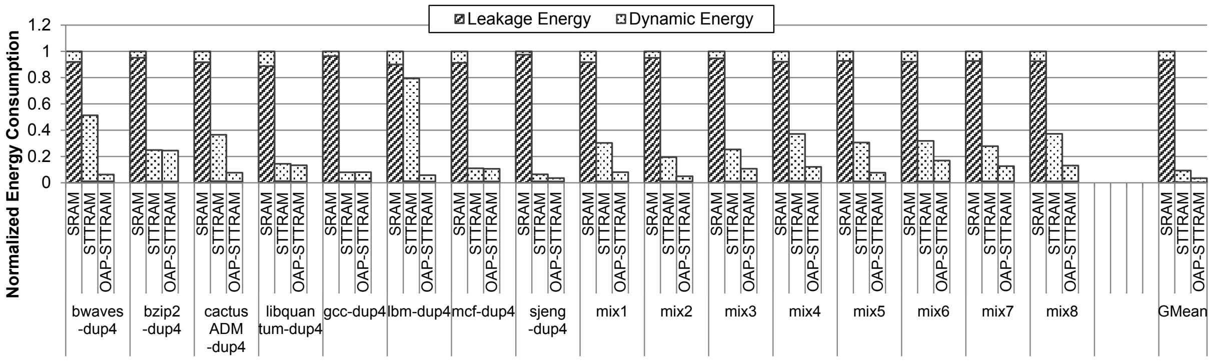

Fig. 7. The energy consumption of SRAM L3, conventional STT-RAM L3 and OAP STT-RAM L3 (normalized to the value of SRAM L3). Each value is broken down to leakage energy and dynamic energy.

technique to identify the access pattern and prevent blocks with a distant reference interval from polluting the cache [21]. However, these work 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.

VI. 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 has higher density and smaller leakage energy, STT-RAM has much longer write latency compared to SRAM, and may cause severe performance degradation in a multi-core system where cache accesses are obstructed by the write operations. In this paper, we proposed OAP, an obstruction-aware cache management policy, which can significantly improve system performance and reduces energy consumption across different workloads for future STT-RAM last-level cache design.

References

[1] K. Nomura *et al.*, “Ultra low power processor using perpendicular-STTMRAM/SRAM based hybrid cache toward next generation normally-off computers,” in *MMM*, 2011.

[2] A. Driskill-Smith, “Latest advances and future prospects of STT-RAM,” in *NVMW*, 2011.

[3] X. Dong *et al.*, “Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement,” in *DAC*, 2008.

[4] G. Sun *et al.*, “A novel 3D stacked MRAM cache architecture for CMPs,” in *HPCA*, 2009.

[5] T. Kawahara *et al.*, “2Mb spin-transfer torque RAM (SPRAM) with bit-by-bit bidirectional current write and parallelizing-direction current read,” in *ISSCC*, 2007.

[6] K. Tsuchida *et al.*, “A 64Mb MRAM with clamped-reference and adequate-reference schemes,” in *ISSCC*, 2010.

[7] S. Thoziyoor *et al.*, “A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies,” in *ISCA*, 2008.

[8] C. H. Kim *et al.*, “A forward body-biased low-leakage SRAM cache: device and architecture considerations,” in *ISLPED*, 2003.

[9] X. Dong *et al.*, “NVSim: A circuit-level performance, energy, and area model for emerging non-volatile memory,” *IEEE TCAD*, vol. 31, no. 0, 2012.

[10] H. M. Rao *et al.*, “Multi-port non-volatile memory that includes a resistive memory element,” US Patent 2011/0 228 594 A1, 09 22, 2011.

[11] M. K. Qureshi and Y. N. Patt, “Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches,” in *MICRO*, 2006.

[12] Y. Xie *et al.*, “PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches,” in *ISCA*, 2009.

[13] N. Binkert *et al.*, “gem5:A Multiple-ISA Full System Simulator with Detailed Memory Model,” in *Computer Architecture News*, 2011.

[14] SPEC CPU, “SPEC CPU2006,” http://www.spec.org/cpu2006/.

[15] W. Xu *et al.*, “Design of Last-Level On-Chip Cache Using Spin-Torque Transfer RAM,” in *IEEE Trans. on VLSI System*, 2011.

[16] C. Smullen *et al.*, “Relaxing Non-Volatility for Fast and Energy-Efficient STT-RAM Caches,” in *HPCA*, 2011.

[17] Z. Sun *et al.*, “Multi Retention Level STT-RAM Cache Designs with a Dynamic Refresh Scheme,” in *MICRO*, 2011.

[18] P. Zhou *et al.*, “Energy reduction for STT-RAM using early write termination,” in *ICCAD*, 2009.

[19] S. Kim *et al.*, “Fair cache sharing and partitioning in a chip multiprocessor architecture,” in *PACT*, 2004.

[20] S. Srikantaiah *et al.*, “Sharp control: Controlled shared cache management in chip multiprocessors,” in *MICRO*, 2009.

[21] A. Jaleel *et al.*, “High performance cache replacement using re-reference interval prediction (RRIP),” in *ISCA*, 2010.