Optimizing MLC-based STT-RAM Caches by Dynamic Block Size Reconfiguration

Jianxing Wang, Pooja Roy, Weng-Fai Wong
School of Computing, National University of Singapore
Republic of Singapore, 117417
{wangjianxing, poojaroy, wongwf}@nus.edu.sg

Xiuyuan Bi, Hai (Helen) Li
Swanson School of Engineering, University of Pittsburgh
Pittsburgh, PA 15261, USA
{xib5, hal66}@pitt.edu

***Abstract*—The use of STT-RAM as on-chip caches has been widely studied. However, existing works focused mainly on single-level cell (SLC) design while the potential of multi-level cell (MLC) STT-RAM has not yet been fully explored. It is expected that MLC STT-RAM can achieve $2\times$ the storage density of SLC and thus improves system performance. Unfortunately, at the device level, the two-step read/write scheme introduces performance and energy overhead. In this paper, we propose an architectural design to dynamically reconfigure the cache block size for a MLC STT-RAM last-level cache. Our approach place certain hot data chunks in smaller blocks so as to benefit from the lower latency and energy, while keeping the rest in larger blocks to maintain an overall hit rate. Experiment shows that our strategy reduces the performance and energy penalty of MLC STT-RAM caches with a slightly higher miss rate. On average, IPC is increased by 4.6% while energy consumption is reduced by 23.5% compared to the conventional MLC STT-RAM cache.**

I. Introduction

A significant portion of the silicon area in any modern processor is occupied by on-chip cache memories. Traditionally, these are implemented with SRAM technology. However, the issue of power consumption is severely hindering the further development of SRAM. As CMOS continues to scale, the shrinking of feature size inevitably increases energy consumption and heat density. It is even predicted that the portion of transistors that can not be turned on simultaneously because of the restricted power budget and cooling capability will soon dominate the chip area [7]. This has prompted researchers to actively look for alternative solutions.

In recent years, *spin-transfer torque* RAM (STT-RAM) has received significant attentions from both the device and architecture communities [5]. It is a *non-volatile* memory (NVM) technology with near-zero standby power, high density, nanosecond-level access speed and CMOS compatibility. All of these advantages make it a promising candidate for constructing large on-chip cache targeted at both high performance and low energy consumption.

Current studies about STT-RAM focused on the *single-level cell* (SLC) design mainly because of its simplicity. On the other hand, STT-RAM inherently can be made to work as *multi-level cell* (MLC), which further increases the storage density [16], [4], [3]. Such a potential has yet to be fully explored due to the multi-step access scheme. Compared to SLC, a typical 2-bit MLC STT-RAM introduces an additional data sensing stage for reads and a two-step programming process for writes. At the architecture level, such an overhead can potentially degrade system performance, reducing the benefits from an enlarged capacity.

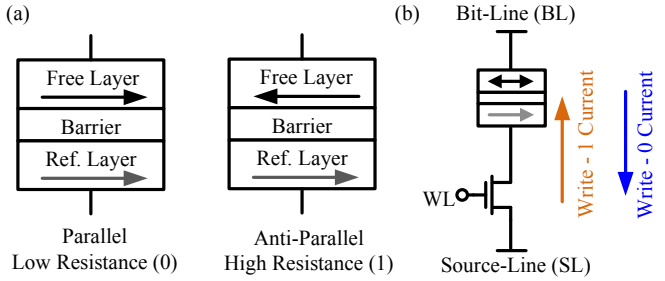

Fig. 1. (a) MTJ in parallel and anti-parallel states; (b) SLC STT-RAM design.

On the other hand, the trade-off between capacity and speed is widely present in many applications. For example, the access to some portions of an application's data may be well spread out while in some portions the accesses are concentrated in only certain hot blocks. Thus, those portions with more uniform access behavior will most likely benefit from a larger cache capacity with higher hit rates while those portions with concentrated accesses will possibly benefit from a lower access latency to a few hot blocks. *Hybrid cache* is such a technique that utilizes the access non-uniformity to mitigate the long write latency incurred from STT-RAM while enjoying an enlarged capacity at the same time [13], [14], [15].

In this paper, we propose an architecture level design for MLC STT-RAM last level caches that is able to satisfy the access requirements of different portions of the data set within arbitrary applications. We utilize an important property of MLC STT-RAM where the *hard-bit* can be turnoff to enable faster and energy-efficient access of the *soft-bit*. For certain cache sets that are more latency-sensitive, only the soft-bits will be used to accelerate access speed while the rest will fully utilize the capacity so as to enjoy a higher hit rate. Unlike previous design where parts of the tag array might be wasted [1], our method does not sacrifice associativity. Experiment shows that it improves overall performance with a much lower energy consumption.

The remainder of this paper is organized as follows. Section II introduces the background of STT-RAM. Section III describes the details of our design. Section IV discusses the experiment results. Section V summarizes related works. This is followed by the conclusion in section VI.

II. Background of STT-RAM

In STT-RAM, data is encoded by the resistance of the *magnetic tunneling junction* (MTJ). An MTJ consists of three layers as shown in Fig. 1(a): two ferromagnetic layers, namely,

978-1-4799-6492-5/14/$31.00 ©2014 IEEE

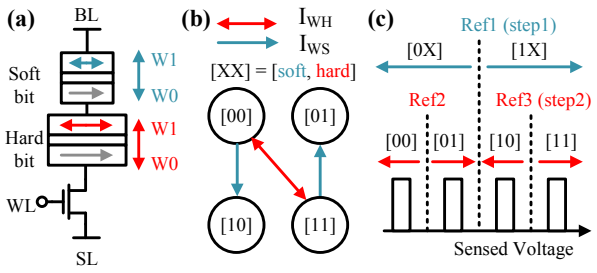

Fig. 2. Serial MLC STT-RAM design. (a) The conventional structure; (b) 2-step write operation; (c) 2-step read operation.

the reference and free layer, separated by a tunnel barrier. The relative magnetization direction of the two ferromagnetic layers determines the resistance of the MTJ. When they are in *parallel* (P) or *anti-parallel* (AP) state, MTJ shows a low- or high-resistance, representing logic '0' or '1', respectively. A small current can be used to sense the resistance and thereby read out the stored data. For write operation, a spin-polarized current which needs to be larger than the *critical switching current* (I_C) is applied to change the magnetization direction of the free layer. When such a current is applied from the free layer to the reference layer, the MTJ will be switched to P state, and vice versa. Fig. 1(b) illustrates the most popular single level cell STT-RAM design that uses a single MTJ, and can be used to store one bit of data [8].

To further improve the storage density, the MLC STT-RAM has been investigated and fabricated. It can store two or more bits in a single memory cell. MLC STT-RAM can be implemented by placing more than one MTJs in a cell. The well accepted series MLC STT-RAM stacks two MTJ vertically and has been proven to be more robust against process variation than other MLC designs [16]. The two MTJ must be of different sizes in order to differentiate two logic bits in one cell. We name the data stored in the small and large MTJs as *soft-bit* and *hard-bit*, respectively, as shown in Fig. 2(a). In a given magnetic process, both the *resistance-area product* (RA) and the *critical switching current density* (J_C) remain constant, therefore the soft-bit has a larger resistance but requires a smaller switching current I_C than the hard-bit.

Programming an MLC requires two stages as shown in Fig. 2(b). First, a strong current $I_{WH} > I_{C,Hard}$ is applied which will switches both hard-bit and soft-bit. Then in the second stage, a smaller current satisfying $I_{C,Hard} > I_{WS} > I_{C,Soft}$ is used to switch only the soft-bit. Similarly, reading out the data from a MLC STT-RAM requires two steps as well. A reference voltage ref–S is first used to detect the soft-bit. Based on the result, a second reference voltage ref–H or ref–H′ is applied to sense the hard-bit, as shown in Fig. 2(c). Therefore, reading and writing the MLC takes more time than a SLC. Note that if the hard-bit is fixed at a value, reading and writing the soft-bit requires only a single step with smaller current densities. Thus, MLC STT-RAM can operate similarly as SLC to accelerate its access speed by reducing half of the capacity.

III. Architecture Design for MLC STT-RAM Cache with Dynamic Block Size Reconfiguration

This section first describes two typical block-level data mapping methods for MLC STT-RAM cache, then followed by an analysis of set-level access patterns. Based on the insights, an architectural design of mixed block sizes is presented.

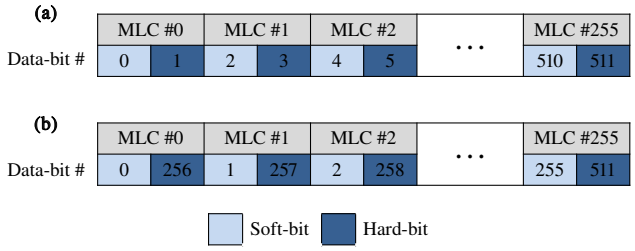

Fig. 3. Block-level data mapping methods for a 512-bit (64-byte) MLC cache block: (a) Direct mapping (DM); (b) Interleaved mapping (IM).

A. Block-level Data Mapping

In using MLC STT-RAM for cache, one of the first design decision is how to organize the data bits within a cache block. For example, given a 512-bit (64-byte) data block and 256 2-bit MLCs, we can store the i-th bit in the $i/2$-th MLC, as shown in Fig. 3(a). We shall refer to this organization as *direct mapping* (DM). DM is a straightforward method without differentiating the accesses to different parts of the block. Therefore, the latency is always the worst-case since both the soft- and hard-bits need to be sensed regardless which data word is accessed.

As an alternative, we can put the lower half (bit 0-255) of the data bits in the soft-region of the MLCs and the higher half (bit 256-511) in the hard-region. This method is illustrated in Fig. 3(b) as shall be referred to as *interleaved mapping* (IM). IM reduces the write latency and energy for the lower half of the block since only one-step programming is required for the soft-bits. However, there is no guarantee that the lower half of the block will endure more writes than the higher half. Thus, the benefits of using such an organization alone is limited.

B. Access Behavior in Last-level Cache

In modern multi-core processors, the access behavior of *last-level* cache (LLC) is usually much more irregular than the upper level caches. The reads and writes are generally distributed non-uniformly across different blocks and sets in the workloads we have studied including solving partial differential equation (PDE), fluid dynamics simulation, real-time raytracing and video encoding. Fig. 4 illustrates four typical access patterns found in PARSEC benchmark suites. We recorded the number of accesses to each half of the block within a cache set. In (a), the lower half of block 0 received substantially more accesses compared to other blocks in the same set. In (b), the lower half of each block has 2× activities compared to the higher half. In (c), most of the access hit on the first few blocks of the set, resulted in the ways being underutilized. The last case is a uniform access pattern where the accesses are spread out across the whole set.

Recall that in the use of MLC STT-RAM, we can either utilize the full 2-bit cell or simply keeping the hard-bit in a fixed resistance and use soft-bit only to accelerate access. In the above examples, (a) and (c) fail to utilize the whole set fully. Thus, a better performance could be obtained by reducing the capacity. In (b), it depends on the difference between the performance gain from a smaller set and the penalty caused by a higher miss rate. For (d), it is most likely that performance will be maximized if the set stays in a larger capacity.

Not only does the cache access pattern behave differently from set to set, but also a set might change its behavior in different program phases. Fig. 5 shows the access pattern

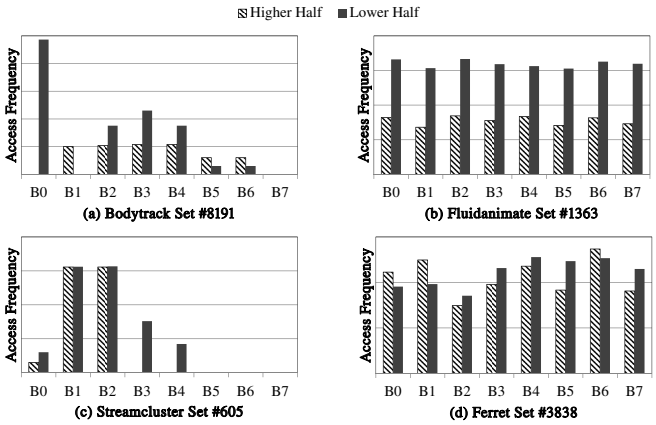

Fig. 4. Various access patterns observed in four benchmarks from PARSEC under an 8-way, 64-byte block, 4MB last-level cache.

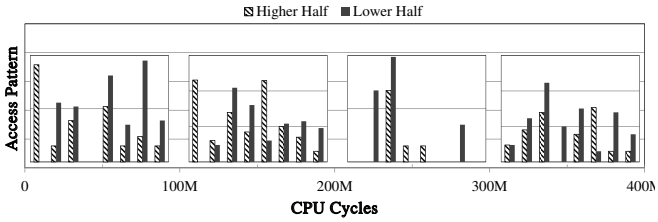

Fig. 5. Access pattern changes in set #129 of *ferret* for the first 400M cycles.

changes in set #129 of *ferret* during the first 400 million CPU cycles. In the first and second phases (0-200M cycles), although block 0, 2 and 3 together are much more active than the others, the overall access pattern tends to spread out and a larger capacity should be maintained. In the third phase (200-300M cycles), the accesses are concentrated in only three half blocks thus latency becomes more critical. Finally, in the last phase (300-400M cycles), the accesses are spread out again. Hence, in the face of such non-uniformities, the optimal system performance could be obtained if each cache set adapts its configuration dynamically response to the access pattern.

C. Dynamic Block Size Reconfiguration

In order to trade capacity for speed, previous work suggested splitting the block-level data mapping for the MLCs and turning off half of the ways to enable faster access [1]. However, this method reduced associativity so thus resulted in a waste of the tag array. Considering the accesses to a large data block are often distributed unevenly between the two halves, as shown in last section, it would be more resource-efficient to dynamically reconfigure the block size rather than the number of ways.

1) Address Decomposition: Under interleaved mapping shown in Fig. 3(b), each cache set can be made to operate in two different modes: a *large block mode* (LBM) and a *small block mode* (SBM). LBM utilizes all the data bits (bit 0-511) in a MLC STT-RAM data block and suffers the maximum access latency. SBM uses only the soft-bits (bit 0-255) and so is much faster and consumes less energy. Fig. 6 illustrates these two operating modes for an 8-way cache set.

While both LBM and SBM can co-exist in a single cache, it must be guaranteed that no data block is mapped to more than one set. Fig. 7(a) describes an address decomposition scheme that supports mixed block modes for a *n*-bit physical address.

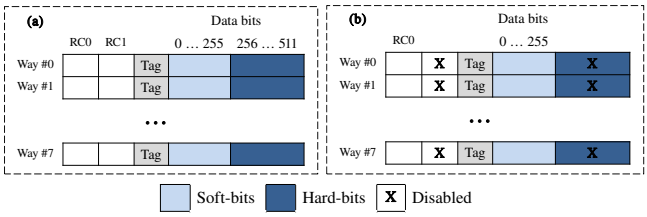

Fig. 6. Two operating modes for cache sets with different block sizes: (a) Large block mode (LBM); (b) Small block mode (SBM).

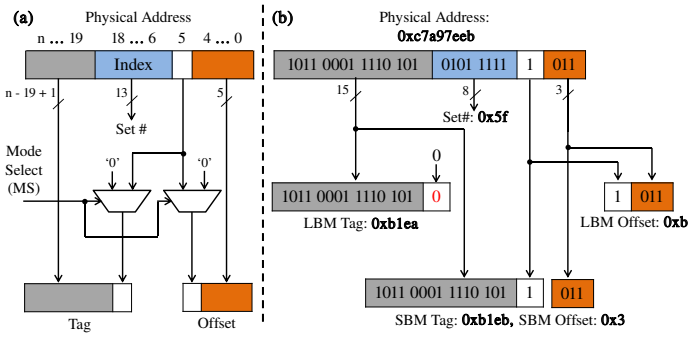

Fig. 7. (a) Physical address decomposition designed for dynamic block size reconfiguration (64-byte block for LBM, 32-byte block for SBM, 8K sets); (b) An example of address decomposition.

The set index is fixed at address bit 6-18 so that each memory location maps to only one cache set regardless of the block size. An additional control bit called *mode selection* (MS-bit) is used to indicate the block mode. When $MS = 0$, the set operates in LBM with a large 64-byte block size. A zero-bit is appended at the end of the tag and the offset is set to the lowest 6 bits of the physical address in order to index any byte within the block.

When in SBM ($MS = 1$), the block sizes are halved at 32-byte. Additional information (bit 5 of the address) is appended at the end of the tag to differentiate the lower and higher halves of a large block. In addition, only a 5-bit offset is now necessary, so the highest bit is zeroed out. Fig. 7(b) illustrates an example of address decomposition for a 32-bit physical address `0xc7a97eeb` which is mapped to the same set `0x5fb` with different tag and offset values under different block operating modes.

2) Reconfiguration Strategy: Ideally, to determine the optimal block operating mode for a particular cache set during a time period, we need to know the number of cache hits and misses it would encounter under each mode. Since such information is not available at runtime, we shall design an algorithm to predict the beneficial operating mode. Here each cache block is equipped with two 2-bit saturating *reference counters*, namely RC0 and RC1 as shown in Fig. 6. Under LBM, RC1 is responsible for the higher half of the block and RC0 is for the lower half while only RC0 will be used in SBM. When a block is initially loaded, its RCs are set to the maximum value (3). When an access results in a hit on the block, it decreases the RC and when it saturates at zero, additional access shall increase all the other RCs within the same set, promoting itself to be a hot (half) block.

When a cache miss occurs, all of the RCs within the set are examined. For a cache set to be switched from LBM to SBM, the number of zeros among all the counters must be less than

a predefined threshold θ_{LS}. For switching from SBM to LBM, the number of zeros shall be larger than another threshold θ_{SL}. This process is shown in algorithm 1.

```
if Cache hit then
    if RC > 0 then
        Decrease it by 1;
    else
        Increase other RCs by 1;
    end
else
    if In LBM and 0 < #Zeros in RCs < θ_LS then
        Reconfigure to SBM;
    end
    if In SBM and #Zeros in RCs > θ_SL then
        Reconfigure to LBM;
    end
end
```

Algorithm 1: Block size reconfiguration policy.

A miss occurring in a LBM set with very few zeros indicates the access to this set are concentrated only certain hot (half) blocks. Therefore, reconfiguring the set to a smaller block size will not introduce significant misses and can possibly obtain performance/energy benefits. On the other hand, if there are too many zeros in a SBM set, each block could have been competing for accesses (Fig. 4 d). Thus, the miss penalty is predicted to be higher than the latency reduction and reconfiguring to LBM might be more profitable.

3) Housekeeping: During reconfiguration, dirty data blocks shall be first written back to memory, followed by a series of housekeeping operations. If the block size is reduced (from LBM to SBM), a simple read-and-write scheme can be adopted to preserve the contents in the lower half block without changing the tag array. This is illustrated in Fig. 8(a). Here we denote the higher and lower half of a large block x as Hx and Lx respectively. The reconfiguration is done by discarding $H0$-$H3$ and resetting bit 256-511.

When going the other way (SBM to LBM), re-fetching the other half for every data block is expensive. Since both the lower and higher halves of a large block may be allocated in the set at the same time, it is necessary to spend extra efforts in tag checking to discover such relationships. Fig. 8(b) shows a case where both the higher and lower halves of block #0 are co-existed in the set. To avoid such checkings, we shall only re-fetch those blocks with 0 in the last bit of the tag and discard all the others. In the above case, $H0$ and $H1$ are written back first if the data are dirty, then followed by re-fetch $H0$ and the unallocated $H2$. Furthermore, since a new data block will be brought in and might result in eviction of an old block, simply by discarding all the data is another choice to reduce the penalties incurred during reconfiguration.

4) Reconfiguration Timing: The overhead of reconfiguration could be hid by choosing a proper timing. For LLC, a miss must be serviced via the main memory. Hence the overhead of RC checkings could be hid inside the miss penalties. However, if the checking is performed during cache hits, the additional latency can no longer be safely ignored. Furthermore, frequent checking might introduce more noises to the prediction. It is therefore more profitable to perform the checking and reconfiguration only during cache miss servicing even though some beneficial reconfigurations might be delayed.

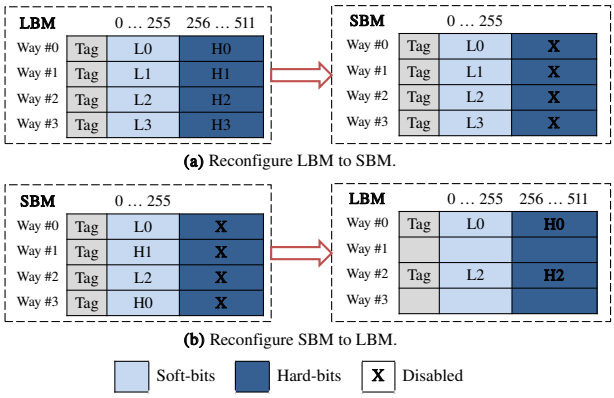

Fig. 8. (a) Example of block size reconfiguration from LBM to SBM in a 4-way cache set; (b) Reconfiguration from the opposite direction.

Key Simulation Parameters	
Processor	2GHz, 4 out-of-order cores
Pipeline	4-way Fetch/Dispatch/Issue/WB/Commit
Functional Units	2 ALUs, 1 FPU, 2 LUs, 1 SU
L1 I- / D-Cache (SRAM)	32KB / 32KB, 4-way, 32-byte block, 2-cycle latency
Coherent Protocol	Illinois MESI
Last-Level Cache (MLC STT-RAM)	8K sets, Maximum 4MB see Table II for the details
Main Memory	4GB, 290-cycle latency

TABLE I. KEY SIMULATION SETTINGS.

5) Other Design Considerations: In order to protect the cache set from harmful reconfigurations, an additional bit is added to each set. When a cache miss occurs, based on the current operating mode and the access counters, the bit will be set to 1 if a reconfigurable access pattern is detected. Only upon the next miss if the same pattern continue to exist will the set be reconfigured. Otherwise, the bit is reset when the access pattern changed during the subsequent miss.

In general, the granularity of mode prediction is controlled by the width of the RC. For a *m*-bit RC, a (half) block requires at least $2^m - 1$ hits to be promoted as a hot (half) block. A larger value of *m* increase the confidence of prediction since it keeps track of more access information. However, it also increases access latency and delays the timing for the beneficial reconfigurations which make the design less effective. A 2-bit RC width is proved to be well effective in the experiment with acceptable performance impact.

IV. EVALUATION

A. Experiment Setup

Our experiments were conducted by using the cycle-accurate simulator MARSSx86 [11]. In particular, we modified the cache controller modules to simulate the architectural level designs. Nine diverse multi-threaded workloads from the PARSEC suite [2] were chosen for the experiment. We executed the benchmarks with *simlarge* input sets. The list of key simulation parameters is given in Table I.

Four different LLC designs are compared:

- **LBM Only**: Baseline MLC STT-RAM cache, the full 2-bit cell is utilized to enable maximum capacity.

LLC Parameters (8K sets, max 4MB capacity)			
Set Operating Mode	LBM	SBM	ASM
Set Configuration	8-way, 64-byte block	8-way, 32-byte block	4-way, 64-byte block
Read Latency (cycle)	10	7	
Write Latency (cycle)	44	23	24
Miss Latency (cycle)		4	
Read Energy (nJ)	0.543	0.419	0.424
Write Energy (nJ)	1.562	0.756	0.853
Miss Energy (nJ)	0.056		0.033

TABLE II. LLC LATENCY AND ENERGY NUMBERS.

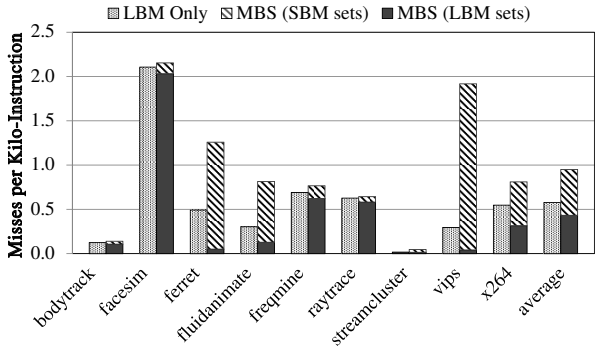

Fig. 9. LLC miss rate comparison.

- **SLC**: We mimic a SLC STT-RAM cache by forcing all cache sets to operate in SBM, thus the overall capacity is halved.
- **Mixed Block Sizes (MBS)**: Our proposed design of mixed block sizes (LBM + SBM) with dynamic reconfiguration. We set both θ_{LS} and θ_{SL} to be 4. By default, all the cache sets operate in LBM.
- **ASM**: The alternative design proposed in [1], where some sets will be dynamically way-reduced to enhance accessing speed. We choose $M_{th} = 512$ as suggested in that paper.

The latency and energy numbers for the last-level cache are generated using NVSim [6] with the MTJ and CMOS technology parameters adopted from [1]. Since NVSim is not able to generate latency and energy numbers for MLC STT-RAM directly, we calculate them by merging two different MTJ designs (hard/soft). Table II shows these parameters for different set operating modes.

B. LLC Miss Rate

Fig. 9 shows the *misses per kilo-instruction* for the LLC of the baseline and our design. Six out of nine benchmarks did not incur noticeable increase of cache misses. In particular, *bodytrack* and *streamcluster* has a very low miss rate since their working set did not exploit the full associativity. In the case like *facesim*, *freqmine* and *raytrace*, the portion of cache sets that was reconfigured to SBM is less than 10% and therefore most misses come from the LBM sets. On the contrary, more than 90% of the sets have been reconfigured to SBM in *ferret* and *vips*, resulted in a significant amount of miss increases.

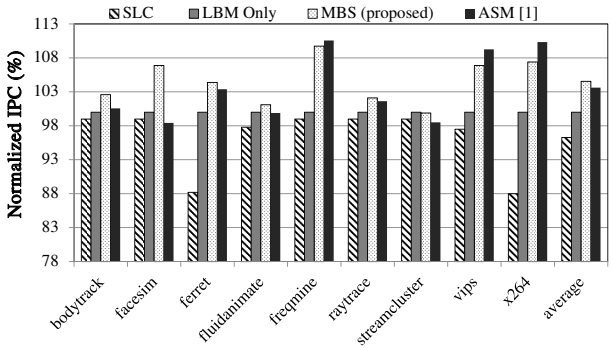

Fig. 10. Normalized IPC comparison.

C. Performance Comparison

Fig. 10 shows the normalized IPC for the designs that have been evaluated. On average, the SLC design with halved capacity incurs about 5% IPC degradation compared to the baseline MLC design, but the effectiveness of using MLC STT-RAM alone varies across different applications. On the other hand, our MBS scheme increases the baseline performance by up to 10%. In particular, *facesim* contains the most number of sets with spread out access behavior and thus result in the lowest frequency of reconfigurations. However, these few number of sets that has been identified to benefit from SBM contribute a rather large increase in IPC (close to 7%). Furthermore, in the case of *ferret* and *vips*, despite that they suffers much larger miss rates, the IPCs are actually 4% and 7% higher. It demonstrates a fact that not all cache accesses are equally performance critical [10]. Our scheme obtains 4.6% of performance improvement on average, 27% more than ASM's improvement over the baseline.

D. Energy Consumption

Since the main leakage source of non-volatile STT-RAM caches is the peripheral circuits which are shared between the three designs, only *dynamic energy* is discussed here. Fig. 11 shows the normalized dynamic energy consumption including those spent in *cache hit*, *miss handling* and additional *housekeeping operations* during reconfigurations. Recall that programming the hard-bit requires much larger current density than the soft-bit, the default LBM Only design is expected to consume more energy than the others. That is because the extra dynamic energy spent in the two-step access is more than the additional miss energy saved from a lower hit rate (SLC, MBS and ASM). The most amount of energy reduction was recorded in *ferret*, where our scheme obtains more than 30% of energy savings. On average, our proposed design consumes 23.5% less energy while ASM consumes 5% more than ours.

E. Sensitivities of θ_{LS} and θ_{SL}

Table. III shows the normalized IPC under different θ_{LS} and θ_{SL} values. Since our simulation adopted an 8-way set-associative LLC setting, the reasonable maximum values of θ_{LS} and θ_{SL} were limited at 8 and 4, respectively. Overall, performance increases as θ_{SL} becomes higher under all θ_{LS} values. This is because most of the cache sets exhibit the same consistent access behavior throughout the application runtime. In another word, when a set is predicted to benefit more from reconfiguration, it is usually the best choice throughout its subsequent life. Hence, we can reduce the chance of reconfiguring the set back to LBM by setting a higher value

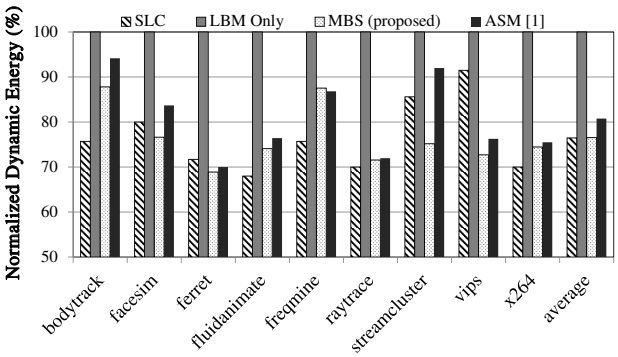

Fig. 11. Normalized dynamic energy comparison.

θ_{LS}\ θ_{SL}	1	2	4
2	97.7%	98.2%	98.35%
4	97.4%	98.65%	100%
6	98.29%	98.4%	98.5%
8	97.8%	98.23%	99.1%

TABLE III. IPC FOR DIFFERENT RECONFIGURE THRESHOLD SETTINGS, NORMALIZED TO $\theta_{LS} = \theta_{SL} = 4$

of θ_{SL}. Based on the experiment results, the optimal values of θ_{LS} and θ_{SL} are found at 4 for both.

V. Related Work

Previous studies on STT-RAM focused on improving the slow and energy-hungry write operation. For instance, Smullen et al. [12] proposed a method of relaxing the non-volatility of STT-RAM to enhance write performance. Sun et al. [14] used different types of STT-RAM cell with mixed retention levels in the cache hierarchy to improve performance and energy consumption. A number of hybrid SRAM and STT-RAM architectures were also proposed [15], [13].

It has been demonstrated that MLC STT-RAM has the potential to further increase the density of conventional SLC STT-RAM. A number of techniques have been investigated to enhance its practicalness. At device/circuit level, *Dichotomic*-based access scheme was proposed to reduce the operation latency [3] while data encoding schemes were examined to improve write energy [4]. At architectural level, Jiang et al. [9] proposed a technique to exploit the difference of R/W characteristics with the construction of two types of cache line. It is also believed that by co-designing at both device and architecture level one can further improve the system performance with less energy consumption of MLC STT-RAM caches [1].

As an emerging technology, MLC STT-RAM also faces reliability problem. Zhang et al. [16] has given a detailed analysis regrading to this issue. While the materials and manufacturing technology continue to improve, eventually MLC STT-RAM shall become mature and practical for actual deployment.

VI. Conclusion

The skewness of cache accesses exists in most if not all applications. In this paper, we exploited such a characteristic in designing energy-efficient caches using emerging memory technology. In particular, we proposed an architecture level design for MLC-based STT-RAM on-chip caches. Our scheme dynamically predicts the access behavior of different cache sets and reconfigures them to appropriate block sizes for better performance and energy conservation. The design neither requires data migration nor wastes the resource of tag array. Simulation showed that compared to the conventional design, our method increased system performance while reduced dynamic energy consumption. As the CMOS technology continues to scale, energy and design complexity issues call for more power-efficient designs. We believe that such a reconfigurable cache built with the emerging STT-RAM technology is an attractive design option for next-generation computing.

References

[1] X. Bi, M. Mao, D. Wang, and H. Li, "Unleashing the potential of mlc stt-ram caches," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 429–436.

[2] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008, pp. 72–81.

[3] Y. Chen, X. Wang, W. Zhu, H. Li, Z. Sun, G. Sun, and Y. Xie, "Access scheme of multi-level cell spin-transfer torque random access memory and its optimization," in *Proceedings of the 53rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2010, pp. 1109–1112.

[4] Y. Chen, W. Wong, L. Hai, and C. Koh, "Processor caches built using multi-level spin-transfer torque ram cells," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2011, pp. 73–78.

[5] Z. Diao, Z. Li, S. Wang, Y. Ding, A. Panchula, E. Chen, L. Wang, and Y. Huai, "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory," *Journal of Physics: Condensed Matter*, vol. 19, no. 16, 2007.

[6] X. Dong, C. Xu, Y. Xie, and N. Jouppi, "NVSim: a Circuit-Level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[7] H. Esmaeilzadeh, E. Blem, R. St Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 365–376.

[8] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto *et al.*, "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram," in *IEEE International Electron Devices Meeting (IEDM)*, 2005, pp. 459–462.

[9] L. Jiang, B. Zhao, Y. Zhang, and J. Yang, "Constructing large and fast multi-level cell stt-mram based cache for embedded processors," in *49th Design Automation Conference (DAC)*, 2012, pp. 907–912.

[10] S. Khan, A. R. Alameldeen, C. Wilkerson, O. Mutlu, and D. A. Jiménez, "Improving cache performance by exploiting read-write disparity," in *Proceedings of the 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[11] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: a full system simulator for multicore x86 CPUs," in *Proceedings of the 48th Design Automation Conference (DAC)*, 2011, pp. 1050–1055.

[12] C. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *Proceedings of the 17th International Symposium on High Performance Computer Architecture (HPCA)*, 2011, pp. 50–61.

[13] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A novel architecture of the 3D stacked MRAM l2 cache for CMPs," in *Proceedings of the 15th International Symposium on High Performance Computer Architecture (HPCA)*, 2009, pp. 239–249.

[14] Z. Sun, X. Bi, H. H. Li, W. Wong, Z. Ong, X. Zhu, and W. Wu, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011, pp. 329–338.

[15] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*, 2009, pp. 34–45.

[16] Y. Zhang, L. Zhang, W. Wen, G. Sun, and Y. Chen, "Multi-level cell stt-ram: Is it realistic or just a dream," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 526–532.