Feasibility Exploration of NVM based I-Cache through MSHR Enhancements

Manu Komalan*†, José Ignacio Gómez Pérez*, Christian Tenllado*, Praveen Raghavan†,
Matthias Hartmann† and Francky Catthoor†
*Complutense University of Madrid (UCM),
Department of Computer Architecture and Automation, Madrid 28040, Spain
†imec, Leuven 3001, Belgium

***Abstract*—SRAM based memory systems are plagued by a number of problems like sub-threshold leakage and susceptibility to read/write failure with dynamic voltage scaling schemes or low supply voltage. Non-Volatile Memory (NVM) technologies are being explored extensively nowadays to replace the conventional SRAM memories even for level 1 (L1) caches. These NVMs like Spin Torque Transfer RAM (STT-MRAM), Resistive-RAM (ReRAM) and Phase Change RAM (PRAM) are less hindered by leakage problems with technology scaling and consume lesser area. However, simple replacement of SRAM by NVMs is not a viable option due to their write related issues. The main focus of this paper is the exploration of write delay and write energy issues in a NVM based L1 Instruction cache (I-cache) for an ARM like single core system. We propose a NVM I-cache and extend its MSHR (Miss Status Handling Register) functionality to address the NVMs write related issues. According to our simulations, appropriate tuning of selective architecture parameters can reduce the performance penalty introduced by the NVM (∼45%) to extremely tolerable levels (∼1%) and show energy gains up to 35%. Furthermore, on configuring our modified NVM based system to occupy area comparable to the original SRAM-based configuration, it outperforms the SRAM baseline and leads to even more energy savings.**

I. Introduction

Memories are increasingly dominating system on chip designs in terms of chip area, performance, power consumption and manufacturing yield. The power analysis of processors reveal sizeable energy consumption by on-chip memories. Reducing the energy consumption of the I-cache organization is vital for reducing the overall energy consumption of the system. Significant effort and resources have been spent on developing emerging memory technologies like ReRAM, Ferro-electric RAM (FeRAM), STT-MRAM and PRAM in recent years. Owing to their desirable characteristics like low leakage power and high density, these NVMs are explored as efficient alternatives for SRAM in scratch-pad and cache memories. However, most of these proposals consist of them being utilized along with SRAM in order to limit the negative impacts (write related issues) and maximize the positive ones.

STT-MRAM, PRAM and ReRAM are some of the more promising NVMs leading the race to replace SRAM. These NVMs exhibit similarities with regards to the asymmetry between their read and write processes. Their write energy consumption and latency is markedly larger than the read counterparts. STT-MRAM has high density, lower power consumption, good performance, and very good endurance (lifetime $\geq 10^{16}$ cycles [1]). ReRAM is also an attractive prospect due to a number of factors like large R ratio, fast read access times, small read energy consumption and area requirement. ReRAM and STT-MRAM technology are CMOS logic compatible and can be integrated along with SRAM on chip. PRAM, however, faces problems in integration and its relatively high read latency puts it at a disadvantage when the focus is on lower level caches. Both ReRAM and PRAM are also plagued by severe endurance issues (lifetime $< 10^{12}$ cycles). Thus, we will only focus on STT-MRAM as the NVM of choice in this paper.

Despite the low energy and high-speed random read access, STT-MRAM writes are hindered by long latency and relatively high write energy. This fails to make simple NVM I-caches feasible. In this paper, we explore the replacement of the traditional SRAM L1 I-cache (IL1) by a NVM one. We propose a novel I-cache configuration that addresses the high write latency and energy issues of the STT-MRAM by means of enhancements to the I-cache MSHR. Since the focus is on system level changes, we will not delve deeply into technology and circuit related matters. This paper will focus on single core ARM like general purpose platforms.

The paper is organized as follows: Section II motivates the need for architectural modifications in order to replace the SRAM I-cache by a NVM counterpart. Section III details the proposed architectural modifications to deal with the performance penalty due to the NVM's long write latencies. Section IV presents the results of the proposed methodologies, and an exploration of the effects of different tunable parameters. Section V discusses similar work and recent developments with respect to the use of NVMs. Section VI concludes the paper.

II. Replacing SRAM with STT-MRAM

The exponential increase of leakage in CMOS transistors with technology scaling poses a big challenge for SRAM memories in the near future [2]. On the other hand, NVMs exhibit better scaling properties along with competitive read latencies, which makes them attractive alternatives to SRAM.

Table I compares the 32KB SRAM and STT-MRAM caches as computed per NVSim [3] for the 32nm tech node

This work has been funded by the Spanish Government's research contract (TIN 2012-32180). Manu Komalan is a recipient of the FPU (Formación del Profesorado Universitario) grant awarded by the Spanish Ministry of Education.

978-3-9815370-2-4/DATE14 © 2014 EDAA

TABLE I. SRAM VS STT-MRAM BASED 32KB I-CACHES VIA NVSIM, [4] AND [5] (32NM TECH NODE, HP TRANSISTORS).

Parameters	SRAM	STT-MRAM
Read energy	0.051nJ	0.037nJ
Write energy	0.049nJ	0.729nJ
Read latency	0.232ns	1.587ns
Write latency	0.176ns	10.261ns
Leakage	121.259mW	31.095mW
Area	$0.315mm^2$	$0.167mm^2$

with High Performance (HP) transistors. The STT-MRAM numbers are obtained by calibrating the NVM array based on models proposed in [4] and [5] via the NVSim tool and appropriate technology scaling for relatively pessimistic scenarios. The NVSim tool, like CACTI [6], is not optimized for small caches and has to be tweaked/modified appropriately to incorporate the cell and array characteristics in [4] and [5]. The STT-MRAM cell access type is CMOS-based and the interface is SRAM compatible.

One of the problems of NVMs like STT-MRAM is the asymmetry in the latencies of their read and write operations. As seen in table I, the write latency is significantly larger than the read counterpart. A simple simulation can show that this asymmetry has a major impact on performance when NVMs are used in the first levels of the memory hierarchy, even for I-caches with low write frequencies. For instance, we have analyzed the impact of a NVM I-Cache on the performance of a Single Core 1GHz ARM processor with 32KB IL1 cache, 64KB DL1 cache and a 2MB unified L2 cache. We use the microarchitecture simulator Gem5 [7] for this analysis and assume that we can maintain the read access time of the SRAM cache (same cycles), while write operations are assumed to range from 10 to 20 cache cycles (1 cache cycle = 2 global/processor clock cycles (2ns) for the 1GHz ARM processor in our GEM5 framework).

Figure 1 shows the performance penalty on replacing just the SRAM I-cache by a NVM counterpart with similar characteristics (size, associativity..). Data cache and the unified L2 remain SRAM based. We ran the simulations with a subset of the SPEC CPU2006, MediaBench and DSPStone benchmarks. For any write latency considered, we observe a clear and unacceptably large performance overhead compared with the baseline. Indeed, *djpeg* may suffer up to 45% performance penalty if the NVM I-cache write latency is as high as 20 cycles. The main conclusion of this analysis is that although STT-MRAM is a good candidate to replace SRAM I-caches, a simple drop-in replacement is not advisable and architecture modifications are required to reduce the impact of their large write latencies.

III. EXTENDED MSHR TO ADDRESS NVM LIMITATIONS

Lockup-free caches are fairly common in modern processors. The first such non-blocking cache, proposed by Kroft [8] utilized a set of registers called MSHR (Miss Status Holding Registers). A simple MSHR is shown in Figure 2(a). Whenever a cache miss occurs a free register from the MSHR is looked up, and its valid bit (V) is set to 1. If there is no free register left, the cache gets blocked and the processor stalls. Otherwise, the memory block of the missing request is annotated in the *block address* field of the selected register, enabling the tracking of further misses to the same block, and the execution

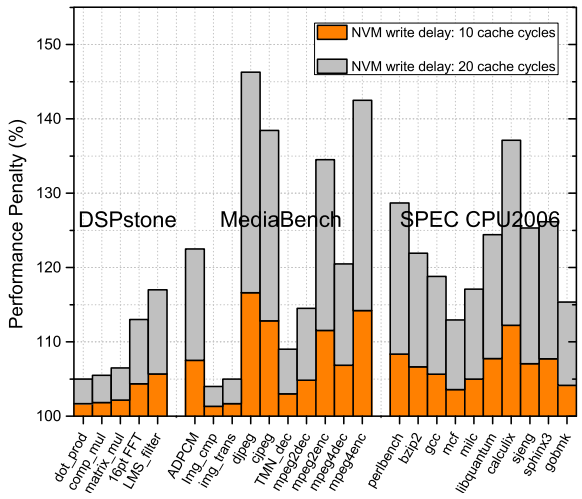
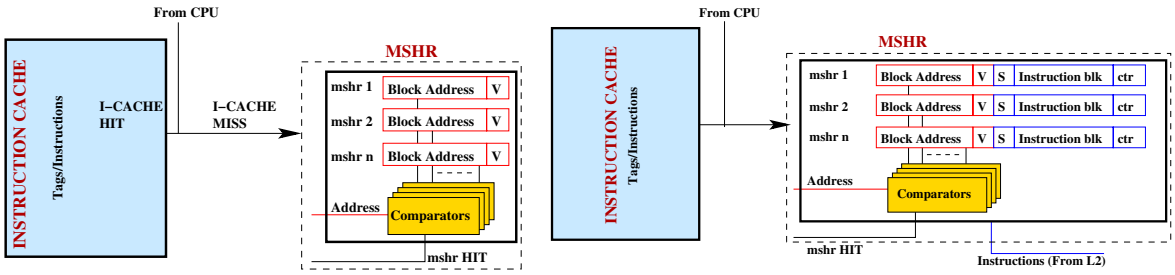


Fig. 1. Performance penalty for the NVM I-Cache, relative to SRAM I-Cache, considering two different ratios of write vs read latencies (delay). SRAM I-cache baseline = 100%.

proceeds. Whenever the miss is served from the next cache level, the requested word is handled by the processor and the cache is conveniently updated. The valid bit (V) is reset to make the entry become free again.

In this paper, we analyze the feasibility of using NVMs for the L1 I-cache, where the only source of write operations are the L2 updates triggered by L1 I-cache misses. We propose using a modified version of Kroft's MSHR organization, in which a small intermediate buffer is used to hold instructions arriving from L2 before they actually update the IL1. This *enhanced* MSHR (EMSHR) is described in figure 2(b). Every register in the EMSHR is extended to hold the complete IL1 block (block size = 128 Bits) corresponding to the memory request it is tracking. In addition, a new control bit 'S' is added to signal that the entry holds valid instructions, i.e. the cache miss was already served and the memory block has been stored in the EMSHR instruction block. Notice that we do not necessarily need to start the transfer of the memory block from the EMSHR instruction block to the NVM IL1 immediately after its arrival from the L2; we can hold it in the EMSHR instruction block to see if there is some reuse. The EMSHR thus effectively acts like a very small (up to 4Kbit: 32 instruction blocks) fully associative cache between the NVM IL1 and the L2 cache. The architecture of the cache also needs to change a little, as the data path now includes the EMSHR. A multiplexer/selector must be included in the modified architecture to allow the processor to select whether to fetch instructions from the STT-MRAM array or the EMSHR.

The fetch sequence of the EMSHR differs slightly from the original MSHR. When the CPU issues a new fetch, the NVM I-cache tag array and the block tag of each register in the EMSHR are checked. If the access hits any of these two structures, the required instruction is served within that access cycle. An EMSHR access can only be considered a hit, if the matching entry has its S bit set. Notice that only one of the two structures would hold the required instruction. If the EMSHR associative search is successful, but the S bit is reset and the entry is valid (V set), there is a previous miss



(a) Original MSHR (b) Extended MSHR

Fig. 2. Original and modified MSHR based cache organization

to the same block. This new *secondary* miss is annotated in the EMSHR and the execution proceeds. Otherwise, a *primary* miss happened and a free register must be allocated. The valid bit and the block request address of the selected register are set. Finally, once the miss is served from the next cache level, the instruction block is written into the allocated register and the S bit is set. Subsequent accesses to this memory block will lead to EMSHR hits.

Two additional aspects still need some special attention. If, no free register is found (all the registers have the V bit set), we must stall the processor if none of them have the S bit set. On the contrary, if one or more registers have the S bit set, one of them must be selected for replacement. In this paper, we rely on the Least Recently Used (LRU) policy. We should remark that in our context no copy-back is needed for any cache line on replacement, because no IL1 line can be dirty.

On the other hand, a *promotion* mechanism must be implemented so that EMSHR entries (occasionally) update the main NVM I-cache. One of our goals is to minimize NVM writes to alleviate performance limitations. Thus, we must carefully select the blocks to be written into the NVM array. We propose to *promote* an EMSHR instruction block whenever the number of accesses to that block reaches a threshold. Therefore, a counter is associated with each register to track the number of references to the block. The rationale behind this proposal is to use the NVM array mainly for loop bodies (which allows code to be repeatedly executed), while mostly sequential code resides in the EMSHR.

Regarding performance, the proposed mechanism decouples the miss from the NVM update, effectively removing the long latency write operation from the critical path. However, those writes may eventually happen and the processor may try to fetch a new instruction while a *promotion* is taking place (since the *promotion* may take as long as 20 cache cycles). There will be no conflict if both operations target different banks in the NVM array. Otherwise, the processor must be stalled, thus degrading system performance.

The EMSHR organization proposed in this section contains several parameters, namely the number of registers, the *promotion threshold* and the NVM array size, that need to be explored to see whether it alleviates the NVM limitations and makes this proposal feasible. These explorations are described in next section.

IV. Exploration and Analysis

In order to evaluate the effectiveness of the proposed modifications, we implement them via the GEM5 simulator([7]). We run all simulations in the System-call Emulation (SE) mode with the *arm_detailed* CPU type that mimics an ARM Cortex-A processor with a clock frequency of 1GHz. The cache configuration is set as follows: a 32KB 2-way associative L1 I-cache, a 64KB 2-way associative L1 data cache and a 2MB 16-way associative unified L2 cache. A single I-cache block is 4 instructions wide (128 bits), so each register in the EMSHR must be able to hold 4 instructions. For simulations, we include a wide range of representative benchmarks from the DSPstone, MediaBench and SPEC CPU2006 benchmark suites.

To evaluate the impact of our proposal on performance, we explored two key EMSHR parameters: *promotion threshold* and EMSHR size (based on the number of registers). Figure 3 highlights these results. From figure 1 (Section II), we know that the performance penalty due to a simple NVM I-cache can reach up to 45%. This can be reduced to 1% penalty with a 32 entry EMSHR (4Kbit size) and a *promotion threshold* of 12. From the figure, it is abundantly clear that our modifications to the I-cache organization helps in reducing performance penalty significantly, to less than 4% for all benchmarks even with a EMSHR capacity of just 8 entries (i.e. 1Kbit).

In figure 3, we see that performance penalty generally reduces for larger EMSHR sizes and higher *promotion thresholds*. Increasing the EMSHR size allows more instruction blocks to be read frequently without accessing the L2 often and thus avoids unnecessary evictions. A reduction in the performance penalty is expected in such a scenario. However, there are physical routing limitations on the extent to which the EMSHR size can be increased and so we limit it to a few KiloBits. Increasing the EMSHR *promotion threshold* means lesser instructions blocks are written into the STT-MRAM I-cache (only the most reused loop bodies are promoted). In most of the bigger benchmarks, the percentage of IL1 capacity misses is very significant, so most code blocks will be evicted before they are fully reused. Given the large write penalty, its better to avoid those writes unless the amount of immediate reuse is large enough even if larger thresholds lead to an increase in L2 accesses.

We also explore the energy consumption variation of the modified system with STT-MRAM I-cache with changes in EMSHR parameters (Figure 4). The energy numbers are obtained via NVSim (Table I) and include the dynamic and static

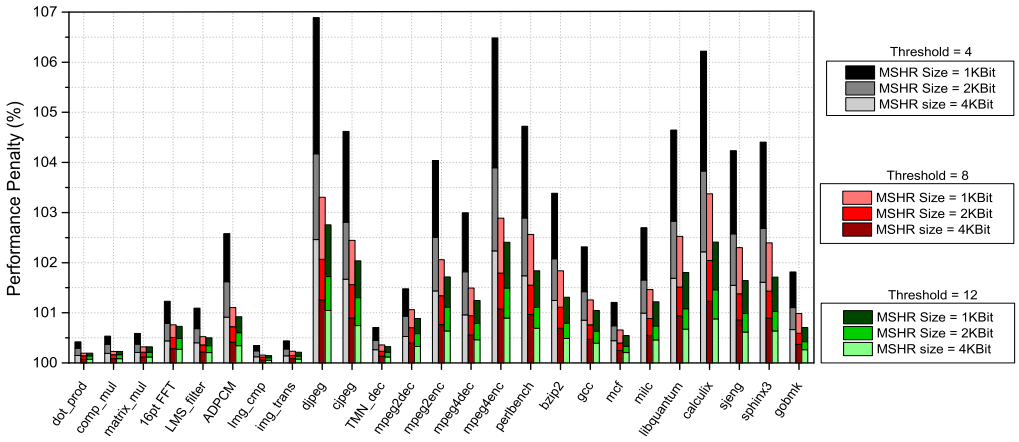


Fig. 3. Performance penalty changes of the STT-MRAM based modified I-cache (relative to the baseline SRAM IMO performance of 100%) for different EMSHR variations.

energy consumption of the L1 I-cache and EMSHR. The read energy consumed on L2 accesses due to L1 I-cache misses is also included here. For all the EMSHR sizes explored, as expected, the larger the EMSHR, lower has been the energy consumption. Thus, if we increase the number of registers, we reduce the number of energy expensive NVM writes without incurring extra L2 accesses.

The *promotion threshold* exploration shows a more interesting behaviour. On increasing the threshold from 4 to 8 accesses, the total energy consumption of all benchmarks reduces. Again, filtering writes to the NVM is globally more energy efficient even at the cost of extra L2 accesses. However, upon further increasing the threshold up to 12 accesses, the largest benchmarks show a clear increase in the energy consumption. Not promoting the loop codes soon enough (and sometimes never if the loop iterations are below 12!) leads to an important increase in the L1 misses. As a consequence, the number of accesses to the bigger L2 increases. Hence, an intricate balance exists between the NVM write energy consumption and the energy consumed by accessing the L2 more often. The small size of the instruction code for the DSP-stone benchmarks make these explorations (both performance and energy) particularly favourable for them compared to the others, since most of the instructions can fit in the EMSHR and there is minimal use of the STT-MRAM array.

To further study the impact of our architectural modifications on the energy consumption of the L1 I-cache organization, we compute and compare the energy breakdown of the STT-MRAM based modified I-Cache organization and the original SRAM based I-cache organization (Figure 5). We fix the MSHR *promotion threshold* and MSHR size to 4 (accesses) and 2Kbit respectively in these simulations. The relative energy consumption for the 2 different scenarios (STT-MRAM based I-cache and SRAM based I-cache) are represented by their respective cumulative bars for each benchmark. The energy consumption of SRAM-based architecture is taken as the baseline here. These bars detail the read, write and static (leakage) energy contributions. Energy contributions from both the L1 I-cache and EMSHR are included. Again, the read energy consumed by means of L2 accesses due to L1 I-cache misses is included.

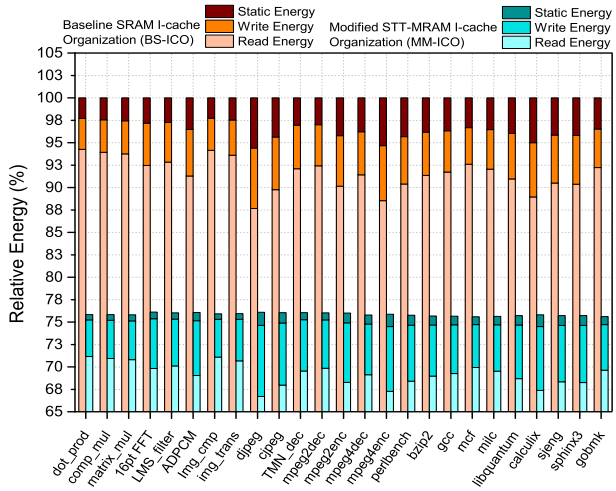


Fig. 5. Energy consumption breakdown of the modified STT-MRAM and original SRAM based L1 I-cache organizations (MSHR size = 2Kbit, threshold = 4). All numbers are relative to the original SRAM based I-cache organization (100%).

As expected, figure 5 shows that the L1 I-cache read energy dominates the total energy consumption. This breakdown helps us to clearly understand which benchmarks had larger read:write ratios, thus fitting better in our NVM I-cache proposal. Certain benchmarks (*mcf, milc, img_trans* etc) have a much larger read:write ratio as compared to the others, which explains its low penalty and energy overhead even when being larger (in dynamic instructions). For the SRAM based I-cache organization, the leakage contribution almost equals that of the write energy contribution for the largest benchmarks. As we move towards lower technology nodes, this contribution will increase significantly and further motivates the transition to NVM based memories. Overall, we can state that the energy consumption of the NVM based I-cache organizations remain fairly competitive relative to the SRAM based I-cache organization, achieving, in average, an 24% reduction for the configuration shown in figure 5.

In all the above experiments we have considered a 32KB STT-MRAM, which is the same size as that of the base-

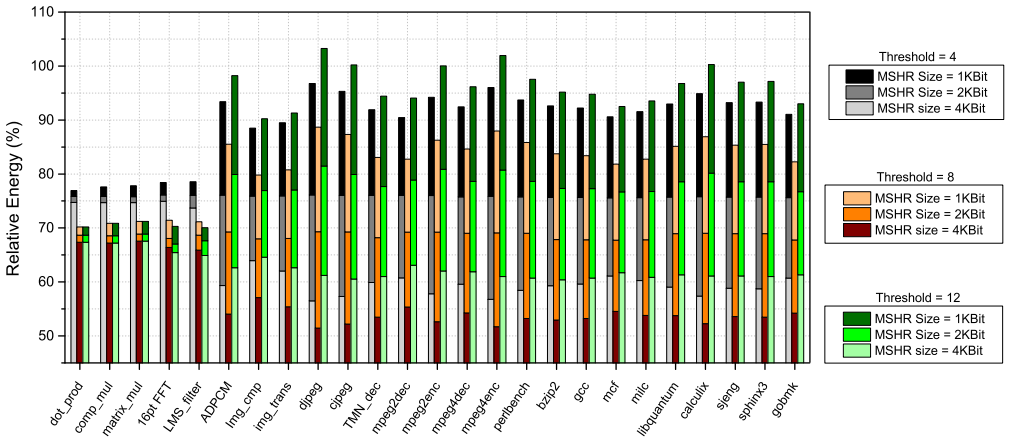


Fig. 4. Energy consumption changes of the STT-MRAM based modified I-cache, relative to the baseline SRAM I-cache (100%) for different MSHR variations.

line SRAM I-cache. The STT-MRAM I-cache is smaller in area when compared to the SRAM I-cache ($0.167mm^2$ vs $0.315mm^2$) due to the much smaller cell size ($14F^2$ vs $146F^2$) even when we take into account the larger transistor to drive higher write current. Hence, we explored the utilization of larger capacity caches. For these experiments we fixed the EMSHR size at 2Kbit. The capacity of the cache be realistically increased by a factor of 2 (64KB STT-MRAM = $0.294mm^2$), taking into account the area overhead of the 2 KBit EMSHR ($0.020mm^2$), and the total area of the proposed memory architecture will remain comparable to the baseline SRAM I-cache area. Figure 6 shows the significant performance improvements achieved with this larger (64KB) STT-MRAM based modified I-cache. There are two important aspects to be highlighted here: firstly, the performance penalty is completely removed and even some minor improvements may be observed. Secondly, with this configuration, the lower the threshold, the better the performance (just the opposite of the behaviour observed in figure 3). The huge reduction in capacity misses makes it much more convenient to promote instruction blocks sooner since they are likely to reside in IL1 much longer than before.

Figure 7 gives the changes in the relative energy associated with this larger (64KB) STT-MRAM based modified I-cache relative to the original SRAM baseline. It's quite clear from this figure and the earlier energy comparisons that a larger cache is clearly beneficial for the larger benchmarks (most of SPEC CPU2006 and Mediabench) and especially those with loop dominated codes provided that the loop iterations are sufficiently large. There is an important reduction in both, the number of writes to the STT-MRAM array and the number of L2 accesses, specially when the promotion threshold gets lower. Only for the smallest benchmarks, where the capacity misses were not a problem, decreasing the threshold does not lead to energy improvements.

V. Related Work

Most of the solutions to the problems faced by replacement of SRAM with NVMs include a combination of software (memory mapping, data allocation) and hardware techniques (registers, buffers, circuit level changes). The reduction of energy consumption by re-directing the data path to use an energy efficient smaller structure like a buffer more frequently has been prevalent for quite some time [9]. However, like it has been shown in our paper, for NVM caches, the aim of such a secondary structure is to filter the writes to the NVM cache and help mitigate their negative impact. The EMSHR, rather than the NVM cache, is the structure that is updated on a I-cache miss in our proposal. [5] presents a similar idea. Here, the MRAM L1 cache is supplemented with several small SRAM buffers to mitigate the performance degradation and dynamic energy overhead induced by MRAM write operations. The data path and the secondary structure differ from our proposal in this case.

[10] proposes a hybrid Scratch Pad Memory (SPM) which consists of a NVM and SRAM. This proposal exploits the ultra-low leakage power consumption and high density of NVM as well as the efficient writes of SRAM. A novel dynamic data allocation algorithm is proposed to make use of the full potential of both NVM and SRAM. For SRAM substitution by NVMs in lower level caches (L2/L3), there have been a number of proposals like [11], [12] and [13]. Unlike the other hybrid architectures presented above, our architecture is primarily a hardware controlled I-cache and makes use of an extended MSHR. The main focus is on ARM like general purpose processing platforms running applications with real-time constraints. Also, while our focus is on the instruction memory, most proposals are data memory specific.

VI. Conclusion

This paper presents a modification of the traditional MSHR organization for the effective exploitation of STT-MRAM based L1 I-caches. Our exploration shows that the appropriate tuning of selective architecture parameters can reduce the performance penalty introduced by the NVM to extremely tolerable levels ($\sim$1%). In addition, we show that we can also obtain significant reductions in energy consumption ($\sim$35%), given that the NVM technology has a favourable read energy per access when compared to SRAM. However, the pareto-optimum values for the different parameters are heavily application and platform dependent. Furthermore, the use of NVMs also allows gains in area and this in turn can be utilized to accommodate I-caches with more capacity ($\sim$2-3 times for STT-RAM). When configuring the modified NVM based I-

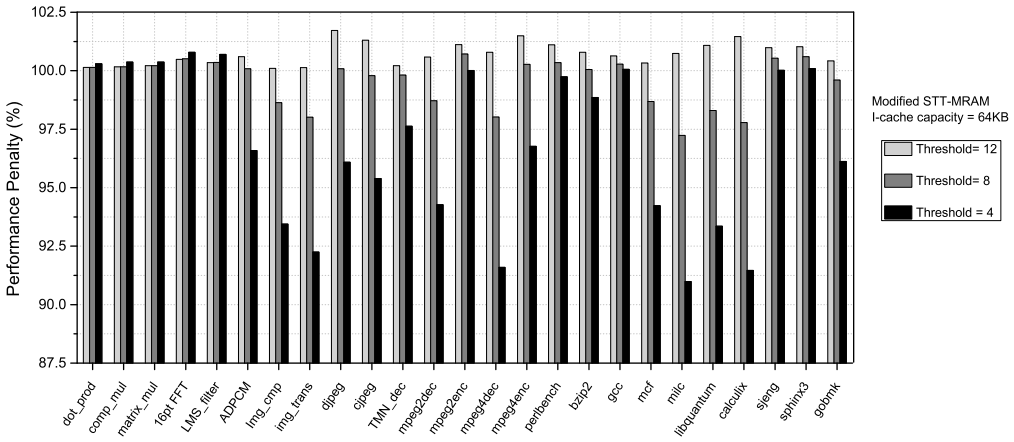


Fig. 6. Performance Penalty variation of the modified STT-MRAM I-cache for different promotion thresholds on increasing the capacity to 64KB. All numbers are relative to the original SRAM based I-cache organization (100%).

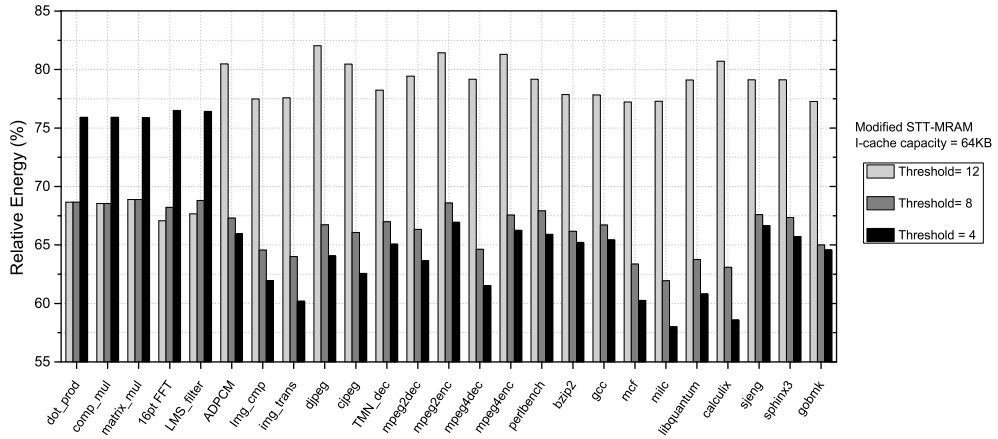


Fig. 7. Energy variation of the modified STT-MRAM I-cache for different promotion thresholds on increasing the capacity to 64KB. All numbers are relative to the original SRAM based I-cache organization (100%).

cache to occupy the same area as the original SRAM-based configuration, the NVM based system outperforms the SRAM baseline and leads to even more energy savings. Finally, the paper shows that STT-MRAM memories are potentially good candidates to build feasible L1 I-caches even for relatively conservative scenarios.

REFERENCES

[1] D. Apalkov *et al.*, "Spin-transfer torque magnetic random access memory (stt-mram)," *J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, pp. 13:1–13:35, May 2013.

[2] G. Shahidi, "Design-technology interaction for post-32 nm node cmos technologies," in *VLSI Technology (VLSIT), 2010 Symposium on*. IBM, 2010, pp. 143–144.

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

[4] X. Guo, E. Ipek, and T. Soyata, "Resistive computation: Avoiding the power wall with low-leakage, stt-mram based computing," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10, 2010, pp. 371–382.

[5] H. Sun, "Using magnetic ram to build low-power and soft error-resilient l1 cache," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 19–28, January 2012.

[6] N. Muralimanohar and R. Balasubramonian, "Cacti 6.0: A tool to understand large caches."

[7] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[8] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," in *Proceedings of the 8th annual symposium on Computer Architecture*, ser. ISCA '81, 1981, pp. 81–87.

[9] A. Janapsatya, S. Parameswaran, and A. Ignjatovic, "Hitme: Low power hit memory buffer for embedded systems," in *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, 2009, pp. 335–340.

[10] J. Hu *et al.*, "Towards energy efficient hybrid on-chip scratch pad memory with non-volatile memory," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2011*. University of Texas, April 2011, pp. 1–6.

[11] P. Mangalagiri *et al.*, "A low-power phase change memory based hybrid cache architecture," in *Proceedings of the Great Lakes symposium on VLSI (GLSVLSI), 2008*. Penn state, March 2008, pp. 395–398.

[12] Y.-T. Chen *et al.*, "Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2012*. Univ. of California, April 2012, pp. 45–50.

[13] A. Jog *et al.*, "Cache revive: Architecting volatile stt-ram caches for enhanced performance in cmps," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, 2012, pp. 243–252.