System level exploration of a STT-MRAM based Level 1 Data-Cache

Manu Perumkunnil Komalan*†‡, Christian Tenllado*, José Ignacio Gómez Pérez*,
Francisco Tirado Fernández* and Francky Catthoor†‡
*Department of Computer Architecture and Automation
Universidad Complutense de Madrid, Spain
†KU Leuven, Leuven 3000, Belgium
‡IMEC, Leuven 3001, Belgium

***Abstract*—Since Non-Volatile Memory (NVM) technologies are being explored extensively nowadays as viable replacements for SRAM based memories in LLCs and even L2 caches, we try to take stock of their potential as level 1 (L1) data caches. These NVMs like Spin Torque Transfer RAM(STT-MRAM), Resistive-RAM(ReRAM) and Phase Change RAM (PRAM) are not subject to leakage problems with technology scaling. They also show significant area gains and lower dynamic power consumption. A direct drop-in replacement of SRAM by NVMs is, however, still not feasible due to a number of shortcomings with latency (write or read) and/or endurance/reliability among them being the major issues. STT-MRAM is increasingly becoming the NVM of choice for high performance and general purpose embedded platforms due to characteristics like low access latency, low power and long lifetime. With advancements in cell technology, and taking into account the stringent reliability and performance requirements for advanced technology nodes, the major bottle-neck to the use of STT-MRAM in high level caches has become read latency (instead of write latency as previously believed). The main focus of this paper is the exploration of read penalty issues in a NVM based L1 data cache (D-cache) for an ARM like single core general purpose system. We propose a design method for the STT-MRAM based D-cache in such a platform. This design addresses the adverse effects due to the STT-MRAM read penalty issues by means of micro-architectural modifications along with code transformations. According to our simulations, the appropriate tuning of selective architecture parameters in our proposal and suitable optimizations can reduce the performance penalty introduced by the NVM (initially $\sim$54%) to extremely tolerable levels ($\sim$8%).**

I. Introduction

SRAM memories nowadays are limited by sub-threshold leakage and susceptibility to read/write failure with dynamic voltage scaling schemes or a low supply voltage. As a result, considerable effort and resources are being utilized in developing emerging memory technologies like ReRAM, Ferro-electric RAM (FeRAM), STT-MRAM and PRAM. Due to their plethora of advantageous characteristics like low leakage, high density and inherent non-volatility, NVMs are being explored as alternatives for SRAM memories even at higher levels of the memory hierarchy like scratch-pad and cache ([1], [2] and [3]). The research on these NVMs has become even more necessary now that memories are increasingly dominating system on chip designs in terms of chip area, performance, power consumption and manufacturing yield. Despite all this, almost all the proposals to incorporate NVMs into the traditional memory hierarchy consists of them being utilized along with SRAM. This is so that the negative impacts (latency and reliability issues being the major ones) can be limited and the positive ones maximized.

STT-MRAM, PRAM and ReRAM are some of the more promising and mature NVM technologies. STT-MRAM is a good candidate to replace conventional SRAM technology for large-size and low-power on-chip caches. STT-MRAM has high density, lower power consumption, good performance (relative to other NVMs and Flash) and suffers minimal degradation over time (lifetime upto 10^{16} cycles [4]). 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 also CMOS logic compatible and can be integrated along with SRAM on chip. PRAM, however, faces problems in integration and its very high write latency puts it at a disadvantage when the focus is on higher level caches. Both PRAM and ReRAM are also plagued by severe endurance issues (lifetime $\leq 10^{12}$ cycles). Thus, we only focus on STT-MRAM as the NVM of choice in this paper and explore the replacement of the traditional SRAM based L1 D-cache (DL1) by a STT-MRAM based one.

Despite the low energy, leakage and very good endurance, STT-MRAM read latency is an issue when we target higher level memories [5]. As a result, a direct drop-in replacement of SRAM by STT-MRAM in the D-caches organization is not feasible. We propose a novel D-cache configuration that is able to overcome the read limitations of the STT-MRAM by means of an intermediate buffer that is termed as the 'Very Wide Buffer' (VWB). This is a significant extension of the design methodology initially proposed in [6] and [7]. We extend the design processes and architectural modifications outlined in both scenarios and come to a solution optimized for our particular specifications. These architectural changes along with appropriate data allocations schemes coupled with code transformations and optimizations are best utilized to make our proposal viable. Since the focus is on system level changes, we will not delve deeply into technology and circuit related matters. To the best of our knowledge, this is the first such work on NVM based L1 D-cache organizations that manages to specifically tackle the read latency limits via micro-architectural modifications and code transformations. This paper will illustrate this on single core ARM like general

This project was partially funded by the Spanish government's research contract: TIN 2012-32180.

purpose platforms, but the principles are not limited to this type of platform.

The paper is organized as follows: Section 2 discusses similar work and recent developments with respect to the use of NVMs. Section 3 motivates the need for architectural modifications in order to replace the SRAM based D-cache by a NVM counterpart. Section 4 details the proposed architectural modifications to deal with the performance penalty raised by the NVMs read latency limits. Section 5 lists the code transformations and optimizations that enable us to fully exploit our architectural modifications and avail the benefits of the NVM cache. Section 6 presents the results of the proposed methodologies, and an exploration of the effects of the different tune-able parameters along with certain comparisons. Section 7 concludes the paper.

II. Related Work

There have been a number of proposals based on hybrid NVM/SRAM organizations for different levels of the memory hierarchy. Almost all of them use a combination of software (memory mapping, data allocation) and hardware techniques (registers, buffers, circuit level changes) to overcome the problems plaguing these proposals. [8] proposes a Hybrid Scratch Pad Memory (HSPM) architecture which consists of SRAM and NVM to take advantage of the ultra-low leakage power, high density of NVM and fast read of SRAM. A novel data allocation algorithm as well as an algorithm to determine NVM/SRAM ratio for the novel HSPM architecture are proposed. In [9], a PRAM based unified cache architecture for L2 caches on high-end microprocessors is proposed and evaluated in terms of area, performance, and energy.

[1] proposes an asymmetric write architecture with redundant blocks (AWARE), that can improve the write latency by taking advantage of the asymmetric write characteristics of 1T-1MTJ STT-MRAM bit-cells. The asymmetry arises due to the nature of the storage element in STT-MRAM, wherein the time required for the two-state transitions (1 to 0 and 0 to 1) is not identical. In [2], 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. [3] 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.

It is quite clear from the work being done in related areas, that NVMs haven't been looked into as options for the highest level of the memory hierarchy very often. Also, unlike the others the main focus here is on bypassing the read latency limitations. Additionally, the write latency oriented techniques do not lead to good results and they do not really mitigate the real latency penalty. Since, the work is based on an ARM like general purpose processing platforms, the latency issues are crucial to the success of the overall system.

III. Replacing SRAM with STT-MRAM: Comparison with existing solutions

The rapid increase of leakage currents in CMOS transistors with technology scaling poses a big challenge for the integration of SRAM memories. This has accelerated the need to shift towards newer and more promising options like STT-MRAM. Table I compares the 64KB SRAM and STT-MRAM caches for the 32nm tech node with High Performance (HP) transistors. The STT-MRAM 64KB D-cache numbers that are used in our experiments are obtained by means of appropriate technology scaling and other optimizations applied to the advanced perpendicular dual MTJ cell with low power, high speed write operation and high magneto-resistive ratio, along with the memory array presented in [5]. These are consistent with the numbers presented by Samsung [10], Toshiba [5], Qualcomm etc. The STT-MRAM cell access type is CMOS-based and the interface is SRAM compatible.

TABLE I. 64KB SRAM L1 D-cache vs 64KB STT-MRAM L1 D-cache

Parameters	SRAM	STT-MRAM
Read Latency	0.787ns	3.37ns
Write Latency	0.773ns	1.86ns
Leakage	204.76mW	28.35mW
Area	$146F^2$	$42F^2$
Associativity	2 way	2 way
Cache Line size	256 Bits	512 Bits

However, like it has been mentioned earlier, latency issues limit the use of STT-MRAM for higher level memories. Previous concerns about STT-MRAM and other similar NVM technologies were along the lines of write-related issues. The read:write latency depends a lot on the R-Ratio (TMR in the case of STT-MAM) in these NVM technologies. With the maturation of the STT-MRAM technology it has become clearer that a high R-Ratio is not realistic, at-least currently, taking into account the cell stability and endurance ([4], [5]: shift from 1T-1MTJ to 2T-2MTJ). Hence, the read latency has become the new major bottleneck to overcome for the substitution of SRAM by STT-MRAM, particularly at the L1 level of the memory hierarchy. As seen in table I, the read latency of STT-MRAM is significantly larger than it's SRAM counterpart.

Write latency issues can still be managed by techniques like the inclusion of a small L0 cache (like in the TMS320C64x/C64x DSP of Texas Instruments [11]), buffers ([2]) or by means of modifications such as the one proposed in [7]. A simple simulation can show that these latency issues, in particular read, have a major impact on performance when NVMs are used in the first levels of the memory hierarchy, even for data caches that are not so read dependent like instruction caches. Here for instance, we have analyzed the impact of a STT-MRAM based D-Cache on the performance of a Single Core 1GHz ARM processor with 32KB IL1, 64KB DL1 and a 2MB unified L2 cache. We utilize the micro-architecture simulator Gem5 [12] for this purpose and realistically assume the read access time of the STT-MRAM cache to be four times that of the SRAM cache, while write access is assumed to be twice that of the SRAM counterpart. We use a subset of the PolyBench Benchmark suite [13] for our simulations.

Figure 1 shows the performance penalty on replacing just the SRAM D-cache by a NVM counterpart with similar characteristics (size, associativity...). The instruction cache and the unified L2 cache remain SRAM based. Even for the

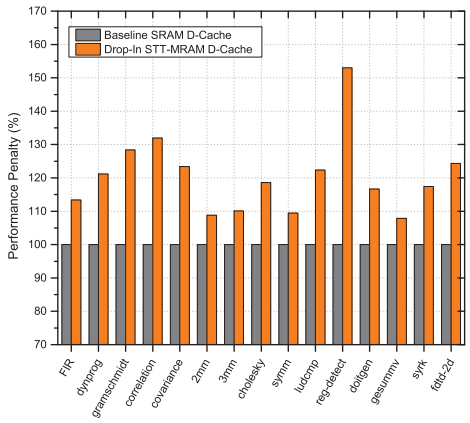

Fig. 1. Performance penalty for the Drop in NVM D-Cache, relative to SRAM D-Cache. SRAM D-cache baseline = 100%.

minimal read latency issue that is being considered here, we can observe a clear and unacceptably large performance overhead compared with the baseline. In-fact, '**reg-detect**' may suffer up to 55% performance penalty if the NVM D-cache is introduced instead of the regular SRAM one. The main conclusion of this analysis is that although STT-MRAM is a good candidate to replace SRAM D-caches given certain obvious advantages, a simple drop-in replacement is not advisable and some architecture modifications are required to reduce the impact of their latency limits.

IV. Architectural Modifications to Address NVM Limitations

In this paper, we analyze the feasibility of using STT-MRAM for the L1 D-cache. We propose using a significantly modified version of a Very Wide Register (VWR) organization focused on improving SIMD access in SRAM caches (proposed in [6]). The enhanced intermediary Very Wide Buffer (VWB) that is proposed here is detailed in figure 2. The Very Wide Buffer here is an asymmetric register file organization. Micro-architecturally, a VWB is made of single ported cells. A post-decode circuit consisting of a multiplexer (MUX) is provided to select the appropriate word(s). The VWB is very close to logic and it can effectively act as a energy efficient high speed buffer between the DL1 and the processor. The interface of this register file organization is asymmetric: wide towards the memory and narrower towards the datapath. This is similar to a register file which is incorporated in the datapath pipeline cycles. The wide interface enables exploiting the locality of access of applications through wide loads from the memory to the VWB and also utilizes this bandwidth to hide latency. At the same time the datapath is able to access words of a smaller width for the actual computations. The wider memory array of the D-cache actually is more beneficial energy wise to the NVM here compared to that of SRAM. This is because the cumulative capacitance of the SRAM transistors across a wide word will be much larger than that of it's NVM counterpart. This has a major effect on the energy and delay of the memory array. The VWB approach can be used in both data-parallel and non-data-parallel contexts.

While the VWB is very small in size, for practical purposes, it has to be able to accommodate at-least 2KBit of data for efficient exploitation of data locality in our latency challenged scenario. The area gains offered by using a NVM make this feasible. The VWB in principle is used to hold critical and frequently executed data arriving from NVM DL1 and exploits data locality when the entire cache line is transferred into it. It must be noted that in the ARM like environment (based on the Cortex A9) that is our platform, in the data cache and instruction cache the word being read out is very wide and usually as big as the entire cache line. Hence, no need is present for any major circuit level modifications in our proposed implementation. The processor will read/write the appropriate word via the MUX selector of a single line from/to the VWB, thereby using a narrow interface matching the data-path of the processor. The STT-MRAM data memory will reads complete lines from the VWB, thereby using a wider interface which will increase the throughput to the memory.

The VWB is modeled like a fully associative buffer. It is made up of two lines of single ported cells in conjunction with each other in such a way that the data can be transferred from one line to another seamlessly back and forth. Both lines are connected to the post decoder MUX network. This way data can be written into and read from the VWB at the same time. Each VWB line has an associated tag. The load and store policies for the L1 data cache and the corresponding VWB in our proposal are as follows:

- Load Operation: The VWB is always checked for the data first during a normal read. On encountering a miss, the NVM DL1 is checked. If the data is present, then it is read from the NVM DL1 and also written into the VWB always. The evicted data from the VWB is stored in the NVM DL1. If the data is not present in the NVM DL1 also, then the miss is served from the next cache level, and the cache line containing the data block is then transferred into the processor and the VWB.
- Store Operation: The data block in the DL1 is only updated via the VWB if it's already present in it. Otherwise, it's directly updated via the processor. A small write buffer is present when to hold the evicted data temporarily, while being transferred to the L2, when the data block in question has to be renewed. No write through is present to the L2 and main memory, and a write-back policy is implemented. If it's a miss, we follow the write allocate policy for the data cache array and a non allocate policy for the VWB. The data in the cache location is loaded in the block from the L2/main memory and this is followed by the write hit operation.

Regarding performance, the proposed mechanism decouples the read hits from the NVM, effectively removing the long latency read operation from the critical path. However, those reads may eventually happen when the VWB encounters a miss and the processor may try to fetch new data while the promotion of a cache line into the VWB is taking place (since the promotion may take as long as 4 cache cycles). We have simulated a banked NVM array, so no conflict will exist if both operations target different banks. Otherwise, the processor must be stalled, thus degrading system performance

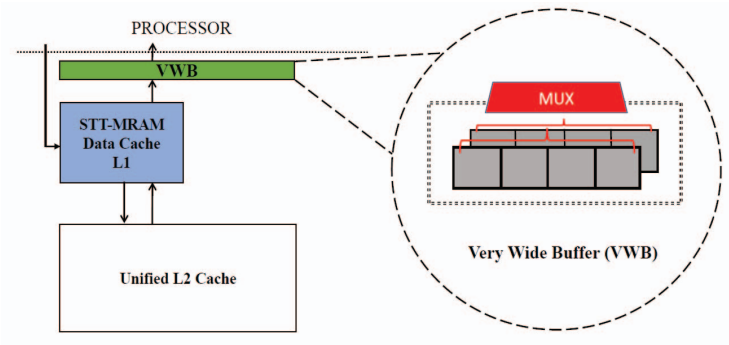

Fig. 2. Modified L1 D-cache organization with a STT-MRAM D-cache and Very Wide Buffer in our General purpose platform ARM like platform.

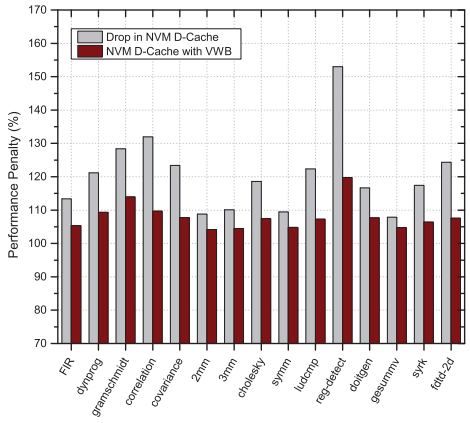

Fig. 3. Performance penalty for the modified NVM D-Cache (with VWB) compared to a simple drop in NVM replacement. SRAM D-cache baseline = 100%.

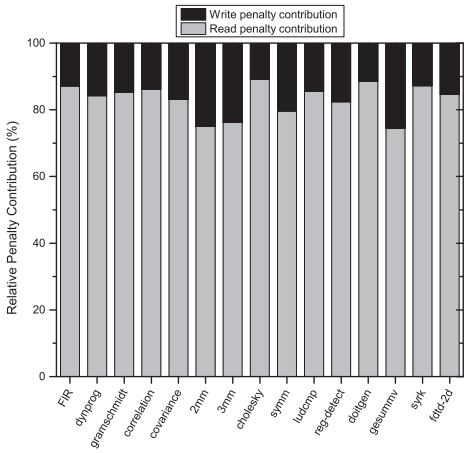

Fig. 4. Performance penalty contribution of the read access latency and write access latency in the our modified NVM based D-cache proposal.

and potentially violating hard real-time operation. For the VWB organization proposed in this section the size and the implementation are the some of the few areas where tweaking to suit the platform, applications and other requirements are possible.

Figure 3 shows the effect of our micro-architectural modifications in reducing the penalty caused by NVM latency limitations. Although the reduction in penalty is significant, it's not not enough taking into account the fact that the benchmarks are minimal in size and aren't really representative of the heavy, robust or data intensive applications that will also be utilized in real time scenario's. Thus we need to cut the penalty further. This can be achieved through the exploitation of parallelization, cutting initial delay time to fetch critical data to the VWB and also cutting the delay by fetching repeatedly used data (in loops) into the VWB. These aspects are described in the next section.

V. Code Transformations and Optimization

Almost all modern systems try to utilize some form of parallelization nowadays for the efficient use of resources and greater performance. We try to exploit data level parallelism here by means of vectorization (essentially loop vectorization). Vectorization is a process wherein the program is converted from a scalar implementation, which processes a single pair of operands at a time, to a vector implementation, which processes one operation on multiple pairs of operands at once. For example, modern conventional computers, including specialized supercomputers, typically have vector operations that simultaneously perform operations such as four additions/subtractions etc. We identify the critical data and loops and vectorize them.

We broke down the contributions of the read and write access to the total penalty of the system for our NVM based proposal to enable us to use appropriate transformations suited to our problem (Figure 4). The read contribution far exceeds that of it's write counterpart towards the total penalty. With increasingly complex kernels, the write penalty contribution also seems to increase, albeit slightly. However, the clear difference in impact between the two even in case of the data cache (as compared to the instruction cache where reads are much more critical) makes a case for the application of pre-fetching. Here, we can pre-fetch critical data and loop arrays to the VWB manually and hence reduce time taken to read it from the NVM. For the moment, we steer these transformations manually by the use of intrinsic functions to modify the individual Kernels.

Alignments of loops, jumps, pointers etc also help in reduction of penalty. We also attempt to transform conditional jumps

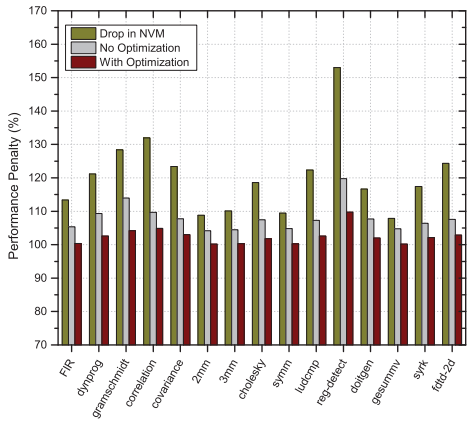

Fig. 5. Performance penalty for the modified NVM DL1 (with VWB) with and without transformations and optimizations. SRAM D-cache baseline = 100%.

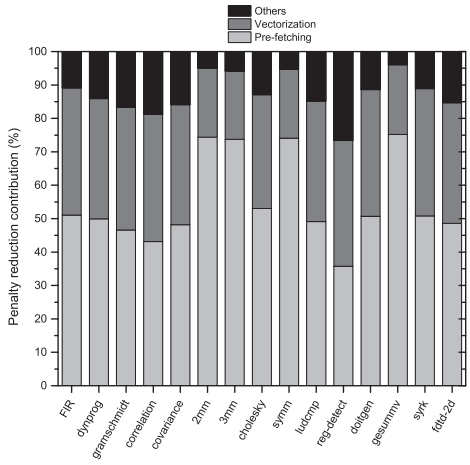

Fig. 6. The contributions of the different transformations to the performance penalty reduction in our proposed NVM DLI (with VWB).

in the innermost loops to branch-less equivalents, guess branch flow probabilities and try to reduce number of branches taken thus improving code locality. We carry out these optimizations automatically by specifying the individual intrinsic function flags at compile time for the different Benchmarks. Figure 5 details the effects of the above mentioned transformations and optimizations on the performance of our modified STT-MRAM based DL1 organization.

Another breakdown of the contribution to reduction of performance penalty, by means of code transformations (Figure 6), reveals that pre-fetching and vectorization have the largest positive impacts. Other intrinsic functions for alignment, branch prediction and avoiding jumps etc become more significant as the kernel becomes larger and more complex. Predictably, pre-fetching is most impactful for the smallest kernels. A systematic approach is being looked into to facilitate and best exploit the above mentioned code transformations and optimizations in an appropriate manner. These will be employed uniformly after further explorations.

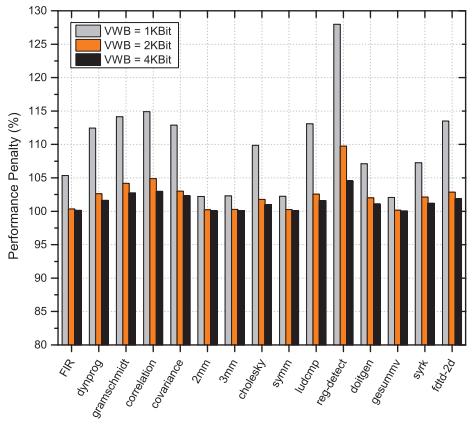

Fig. 7. Performance penalty of the proposal for different sizes of the VWB in our proposal (optimized NVM DL1 with VWB). SRAM D-cache baseline = 100%.

VI. Exploration and Analysis

In order to gauge the effectiveness of the proposed modifications, we implement our proposal and design methodologies via the GEM5 simulator ([12]). We run all simulations in the System-call Emulation (SE) mode with the ARM detailed CPU type that mimics an ARM Cortex-A9 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 D-cache and a 2MB 16-way associative unified L2 cache. For the simulations, we utilize a subset of the PolyBench benchmark suite like it has been mentioned before. Although the set of benchmarks tested here are not particularly large or heavily data intensive, it stands to reason that a fair extrapolation of these conditions even for larger benchmarks would produce significant reduction in the performance penalty induced by the introduction on an NVM cache in Level 1. The VWB size is usually taken to be 2KBit with 2 lines of 1KBit register files. The D-cache line size is 512 Bits.

One of our first explorations looks into the effect of the size of the VWB on our proposed idea (Figure 7). It is quite clear from the figure that larger size VWB's help in reducing the penalty more. This is simply because of more code being able to fit into the VWB with it's increased capacity and hence lesser performance penalty. However, a limit is present to the VWB size put forward by technology, circuit level aspects cost and energy. The routing and layout also becomes cumbersome. Hence, we found it ideal to keep the size of the VWB to around 2KBit considering the area gains offered by the NVM. Keep in mind that a fully associative search also becomes a big problem with the increase in size of the VWB.

For a better idea of where our proposal stands, we compared it to a few techniques for write mitigation in NVMs like a variation of the commonly used L0 cache and the Enhaced MSHR presented in [7]. We utilize these proposals for the purpose of latency reduction in our scenario, where read latency is more of a problem. The hardware structures are made fully associative and have the same size (2KBit) as that of the VWB for a fair comparison. However, the given structures are not as wide as the VWB and conform to the interface of the regular size memory array. The comparison

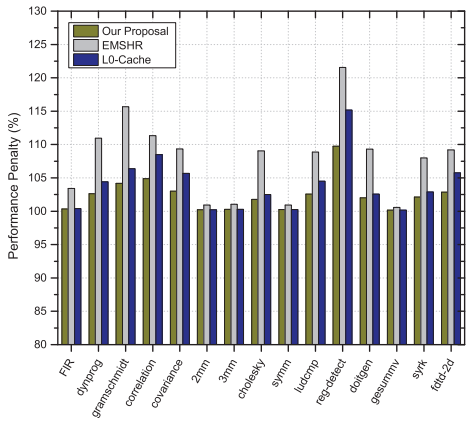

Fig. 8. Performance penalty comparisons between our proposal, a modified version of commonly employed L0 Cache and the EMSHR proposed in [7]. SRAM D-cache baseline = 100%.

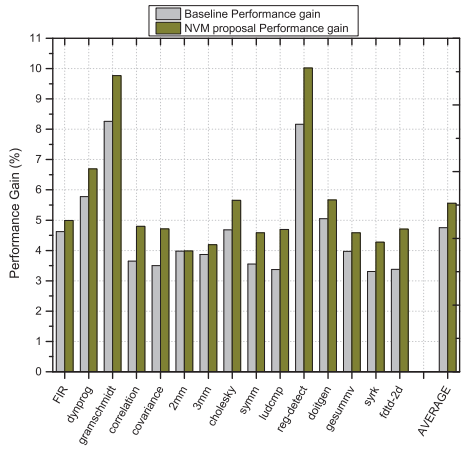

Fig. 9. Effect of code transformations and optimizations on the baseline SRAM D-cache and a comparison with our proposal (NVM DL1 with VWB).

is detailed in Figure 8. Our proposal offers almost twice the penalty reduction as compared to the other previous proposals. This is due to the uniqueness of the structure and the software optimizations included to exploit it.

The effect of code transformations on the baseline SRAM based system is also a net positive. We have compared the net effect of the optimizations and transformations on the baseline system to that of our proposal (Figure 9). It stands to reason that while the software transformations can positively affect the baseline SRAM system (resulting in a better performance compared to our proposal by $\sim$8%), it is more pronounced in case of our NVM based proposal where the architecture and data allocation policy is tuned to exploit these optimizations the most. Also, these do not take into account the obvious advantages offered by the NVM cache like more area leading to more capacity, and lower energy.

VII. Conclusion

This paper presents a modification of the traditional data cache organization by the inclusion of a NVM based cache instead of the traditional SRAM one along with the addition of a Very Wide Buffer for the effective exploitation of this STT-MRAM cache. This Very Wide Buffer also helps with the reduction of the performance penalty induced due to STT-MRAM latency limitations. Our explorations show that appropriate tuning of selective architecture parameters along with suitable data allocation techniques coupled with code transformations and optimizations can reduce the performance penalty introduced by the NVM to extremely tolerable levels (8%) even in the worst cases. Furthermore, the use of NVMs also allows gains in area and even energy (power models have yet to be fully developed though). The gains in area can in turn can be utilized to accommodate D-caches with more capacity (around 2-3 times for STT-MRAM). Although the set of benchmarks tested here are not particularly large or heavily data intensive, it stands to reason that a fair extrapolation of these conditions even for larger benchmarks would result in a significant reduction of the performance penalty induced by the introduction of a NVM DL1.

References

[1] K.-W. Kwon, S. Choday, Y. Kim, and K. Roy, "Aware (asymmetric write architecture with redundant blocks): A high write speed stt-mram cache architecture," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, no. 4, pp. 712–720, April 2014.

[2] 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.

[3] J. Hu, C. Xue, Q. Zhuge, W.-C. Tseng, and E.-M. Sha, "Towards energy efficient hybrid on-chip scratch pad memory with non-volatile memory," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.

[4] 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.

[5] H. Noguchi, K. Ikegami, N. Shimomura, T. Tetsufumi, J. Ito, and S. Fujita, "Highly reliable and low-power nonvolatile cache memory with advanced perpendicular stt-mram for high-performance cpu," in *VLSI Circuits Digest of Technical Papers, 2014 Symposium on*, June 2014, pp. 1–2.

[6] P. Raghavan, A. Lambrechts, M. Jayapala, F. Catthoor, D. Verkest, and H. Corporaal, "Very wide register: An asymmetric register file organization for low power embedded processors," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2007*. IMEC, April 2007, pp. 1–6.

[7] M. Komalan, J. I. G. Pérez, C. Tenllado, P. Raghavan, M. Hartmann, and F. Catthoor, "Feasibility exploration of nvm based i-cache through mshr enhancements," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14, 3001 Leuven, Belgium, 2014, pp. 21:1–21:6.

[8] J. Hu, Q. Zhuge, C. Xue, W.-C. Tseng, and E.-M. Sha, "Optimizing data allocation and memory configuration for non-volatile memory based hybrid spm on embedded cmps," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, May 2012, pp. 982–989.

[9] 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.

[10] A. V. Khvalkovskiy *et al.*, "Basic principles of STT-MRAM cell operation in memory arrays," *Journal of Physics D Applied Physics*, vol. 46, no. 7, p. 074001, Feb. 2013.

[11] *TMS320C64x/C64x+ DSP CPU and Instruction Set : Reference guide*, Texas Instruments, July 2010. [Online]. Available: http://www.ti.com/lit/ug/spru732j/spru732j.pdf

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

[13] L.-N. Pouchet. (2012) Polybench: The polyhedral benchmark suite. [Online]. Available: http://www.cs.ucla.edu/~pouchet/software/polybench/