Exploring Configurable Non-Volatile Memory-based Caches for Energy-Efficient Embedded Systems

Tosiron Adegbija
Department of Electrical and Computer Engineering
University of Arizona
Tucson, Arizona
tosiron@email.arizona.edu

ABSTRACT

Non-volatile memory (NVM) technologies have recently emerged as alternatives to traditional SRAM-based cache memories, since NVMs offer advantages such as non-volatility, low leakage power, fast read speed, and high density. However, NVMs also have disadvantages, such as high write latency and energy, which necessitate further research into robust optimization techniques. In this paper, we propose and evaluate *configurable non-volatile memories (configNVM)* as a viable NVM optimization technique, and show that configNVMs can reduce the cache's energy consumption by up to 60%, with minimal performance degradation. We describe the knowledge gaps that must be filled to enable configNVMs, and show that configNVMs offer new opportunities for energy efficient caching in embedded systems.

CCS Concepts

•Computer systems organization → Embedded systems; Processors and memory architectures; •Hardware → Non-volatile memory;

Keywords

Configurable memory, non-volatile memory, cache memories, low-power design, low-power embedded systems, adaptable hardware.

1. INTRODUCTION AND MOTIVATION

Caches have been the focus of much optimization research, since they significantly impact performance and account for a large percentage of a microprocessor's total system power and energy consumption, especially in embedded systems. A commonly researched cache optimization involves using highly configurable caches [10, 24] whose configurations (cache size, associativity, and line size) can be dynamically adapted to varying runtime application execution requirements. Previous work has shown that such configurable caches can reduce cache energy consumption by up to 62% [11].

GLSVLSI '16, May 18 - 20, 2016, Boston, MA, USA

ISBN 978-1-4503-4274-2/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2902961.2903009

More recently, non-volatile memories (NVMs), such as spin-torque-transfer random access memory (STTRAM) and resistive RAM (ReRAM) have emerged as promising alternatives to the traditional volatile, SRAM-based memories [13]. While NVM-based memories are still nascent, studies have shown that they offer advantages over traditional memories, such as low leakage power, non-volatility, and high density [20]. These advantages make NVM-based technologies especially attractive for caches in resource constrained embedded systems. However, NVMs also result in overheads, such as high write latency, dynamic energy, that necessitate much ongoing research into NVM optimization techniques [14]. For example, previous works [19, 22, 25] have explored architectural-level and circuit-level techniques for masking the effects of high write latencies and write energy. Alternatively, a recent work [13] explored the tuning of data retention time in order to reduce write latencies and write energy.

In this paper, our overarching goal is to design an NVM-based cache that is energy efficient, low-overhead (in terms of performance and area), adaptable to varying application requirements, and easy to implement. Therefore, we propose and draw attention to *configurable NVMs (configNVMs)* as a viable low-overhead optimization technique for energy efficient caches in low-power embedded systems. ConfigNVMs dynamically adapt the cache's configurations to varying application cache requirements, in order to minimize the energy consumption without incurring significant performance overheads. Due to the optimization potential that configurability affords, we motivate research focus on robust and efficient techniques for designing configNVMs, and propose that configurability/adaptability should be an intrinsic design characteristic of NVM technologies, and not an afterthought. We explore the energy benefits of configNVMs and show that by adapting NVM-based caches' configurations to different applications' characteristics, significant energy savings are achievable as compared to non-configurable NVM-based caches. We show that augmenting an NVM-based cache for configurability incurs little design time overhead, and suggest techniques for achieving configNVMs.

The main contributions of this paper are summarized as follows:

• Focusing on embedded systems benchmarks, and using the ReRAM and STTRAM as case studies of promising NVM technologies, we empirically show that configNVMs can reduce average energy consumption by up to 60%, as compared to non-configurable NVMs, without degrading the performance.

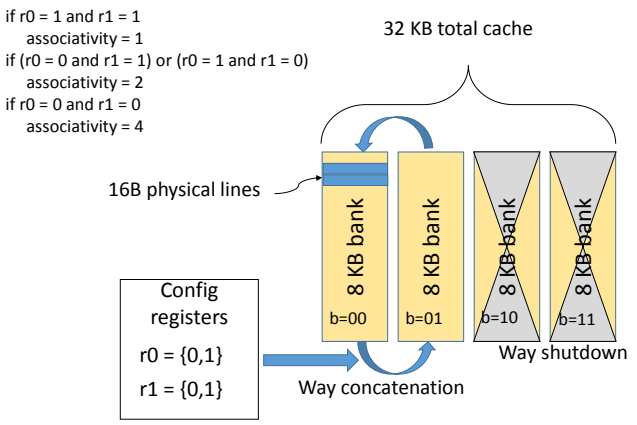

Figure 1: Illustration of configNVM architecture

• We analyze the tradeoffs of STTRAM- and ReRAM-based configNVMs for different applications, and show that the different technologies perform differently for different applications. Therefore, we give insight into which of the two technologies designers should focus on when researching and designing configNVM techniques.
• Drawing from previous work on configurable SRAM-based caches, we describe the knowledge gaps that must be filled in order to enable configNVMs.

2. RELATED WORK

Different circuit-level and architecture-level techniques have been proposed for reducing the high write latency and energy consumption imposed by NVMs. Smullen et al. [18] proposed to reduce the high energy and write latency by lowering the STTRAM's retention time, thereby relaxing the non-volatility. The authors used a combination of SRAM-based L1 caches and reduced-retention STTRAM L2 and L3 caches to eliminate performance loss and reduce the energy-delay product. Rasquinha et al. [15] proposed a microarchitectural optimization that reduced the STTRAM's write energy by preventing the premature eviction of cache lines to lower cache levels. They used a replacement algorithm that increased the residency of dirty lines in the L1 cache in order to accommodate all the stores to those lines, at the expense of higher miss rates.

Xu et al. [23] quantitatively studied the impact of the STTRAM's memory cell sizing on overall computing system performance and showed that different applications have different memory sizing expectations. The authors used the STTRAM for the L2 cache in their studies. Xu et al. [21] studied the memristor-based ReRAM design and provided insight on the different design choices for the best tradeoffs in performance, energy, and area. In this paper we focus on low-power, resource constrained embedded systems, and propose configNVMs as a complementary low-overhead microarchitectural optimization to previous optimization techniques. ConfigNVMs reduce non-volatile memories' energy consumption, while incurring minimal performance degradation. Our major design goals are high energy savings, adaptability to application resource requirements, low performance and area overheads, and ease of design and implementation.

3. ARCHITECTING CONFIGNVMS

Configurable non-volatile memory-based caches (ConfigNVMs) allow the cache configurations to be dynamically adapted to varying runtime application resource requirements. To architect configNVMs, three key design challenges must be addressed: *augmenting NVM caches for configurability*, *cache tuning heuristics/algorithms*, and *low-overhead cache tuners*. Since configurable caches have been well researched in traditional SRAM-based caches, several previous techniques can be reused and/or modified for configNVMs, allowing previous design efforts to be amortized. Furthermore, NVMs generally have similar electrical interfaces to SRAMs [8], thus, configNVMs can also be organized similarly to traditional configurable SRAM-based caches. The most important criterion for techniques used to architect configNVMs is that they most impose minimal area and performance overhead on the system. In this section, we describe our approaches to addressing the configNVM design challenges.

3.1 Augmenting NVM Caches for Configurability

To augment NVM caches for configurability, we use way shutdown, way concatenation, and line concatenation [24] to configure the cache size, associativity, and line size, respectively. In the following discussions, we assume a cache with a maximum of four ways, however, the techniques described herein can be extended to any n-way system.

Figure 1 illustrates the configNVM's architecture. Each configNVM comprises of four banks, with each bank composed of the memory cells (e.g., STTRAM or ReRAM cells). The banks act as ways, and are organized such that different ways can be concatenated and/or shutdown (using power-gating) to configure the configNVM cache configurations. Using two single-bit configuration registers (r0 and r1), way concatenation allows the cache to be configured as one-way/direct-mapped (when r0 = 0 and r1 = 0), two-way (when r0 = 0 and r1 = 1 or r0 = 1 and r1 = 0), or four-way set associative (when r0 = 1 and r1 = 1). Each bank also has a bank ID, b, which specifies the bank that should be gated for way shutdown (Section 3.2). For example, given a 32 KB configNVM as shown in Figure 1, the configNVM comprises of four 8 KB banks, each of which serves as a way, and can be gated to configure the cache size into 8 KB (shutdown three banks), 16 KB (shutdown two banks) or 32 KB (no bank shutdown). When the banks are shut down for 8 KB and 16 KB caches, the value of b determines which specific banks are shutdown, based on the cache tuning heuristic (Section 3.2). Finally, the cache's physical line size is 16 B, and multiple contiguous lines can be fetched to logically configure the line size to 32 B or 64 B.

3.2 Cache Tuning Heuristics/Algorithms

Given a configNVM with several possible configurations, the design space exploration method is critical to mitigating runtime overheads, and necessitates efficient tuning heuristics/algorithms. To enable our experiments and analysis, we developed a simple cache tuning heuristic. Since the goal of this work is to show the benefits of configNVM, we decided to use a simple heuristic for our evaluations, however, we intend to develop more elaborate and robust heuristics/algorithms for future work. Figure 2 depicts our tuning heuristic, which explores different cache configurations and determines the best cache configurations for the different

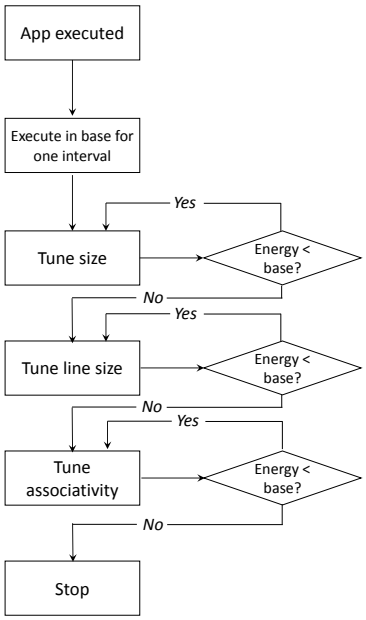

Figure 2: Tuning heuristic

applications based on the energy consumption. Each potential configuration is explored for one tuning interval, which could be measured by the number of executed instruction or in time [17]. For our experiments, we used a tuning interval of 10ms, during which we empirically determined that stable execution statistics can be gathered. Some modern operating systems (e.g., Linux) also use sampling intervals (OS scheduler ticks) of 10ms.

When an application is executed for the first time, the heuristic first executes the application in the largest/base cache configurations for one tuning interval. The heuristic then decreases the cache size as long as decreasing the cache size reduces the energy consumption as compared to the base configuration. Thereafter, the heuristic decreases the line size as long as decreasing the line size reduces the energy consumption, followed by similar adjustments to the associativity. The final configuration is then stored in a configuration table for subsequent executions of that application. Figure 3 depicts the basic structure of the configuration table. The configuration table is a small hardware structure that consists of three fields: *appID*, *config*, and *bankID* fields. The *appID* field stores the application's identification, *config* stores the application's best configuration as determined by our tuning heuristic, and *bankID* stores the identification of the application's most recently used cache bank. In order to potentially reduce the number of compulsory cache misses, when the application is subsequently executed, the application's most recently used banks are reused. Since

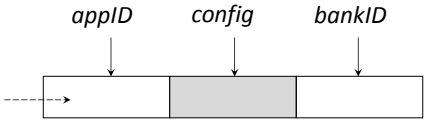

Figure 3: Basic structure of the configuration table

NVM caches retain their data after the banks are shutdown, reusing the banks may allow some of that application's previously fetched memory blocks to be used without having to fetch the blocks from lower levels of the memory hierarchy. However, if the banks are occupied, then alternate banks can be used.

3.3 Cache Tuners

In order to orchestrate the tuning process and implement the tuning heuristics, low-overhead software or hardware cache tuners are required. Software tuners use the system's processor to execute the tuning heuristics, which enables easy system integration. However, software tuners can also affect the cache and runtime behavior due to context switching. These effects can cause the heuristic to choose inferior configurations. Alternatively, hardware tuners alleviate these effects by incorporating custom hardware that execute the heuristics [4]. For our work, we designed a custom hardware tuner using synthesizable VHDL, and quantified the area and power overheads using Synopsys Design Compiler [7]. Relative to an ARM Cortex A9 microprocessor [1], which we used to represent current embedded systems, our tuner imposes 1.2% and 1% area and power overheads, respectively. We omit low level details of the tuner designs for brevity, since that is not the focus of this paper.

4. EXPERIMENTS

In this section, we compare the proposed configNVM with a base NVM cache with fixed configurations for both the STTRAM and ReRAM technologies.

4.1 Experimental Setup

We evaluated the proposed configNVM by comparing a system that switches to the best cache configuration for each executing application to a base system with fixed cache configurations. The base system had 32 KB, 4-way private level one (L1) instruction and data caches with 64 B line sizes, similar to several state-of-the-art embedded systems microprocessors' cache configurations [1]. Thus, the configNVM offered 8 KB, 16 KB, and 32KB cache sizes, 1, 2, and 4-way set associativity, and 16 B, 32 B, and 64 B line sizes. We simulated our work using a combination of the GEM5 simulator [6], to gather cache execution statistics, and NVSim [9] to calculate the energy and latency statistics for the STTRAM and ReRAM technologies. GEM5 is a widely used simulator for computer architecture research, while NVSim is a circuit-level model for NVM performance, energy, and area estimation, which has been validated against industrial NVM prototypes.

To model a variety of embedded systems applications, we used a total of seventeen embedded systems benchmarks, comprising of twelve EEMBC Automotive benchmarks [2] and five Mibench benchmarks [12] that were selected to represent different application domains. We used a tuning interval of 10 ms, and assumed a system with persistent applications that execute several times throughout the system's lifetime. For all the benchmarks, the tuning process completed during a few executions, thus, the tuning overhead amortized very rapidly over subsequent executions (details in Section 4.4).

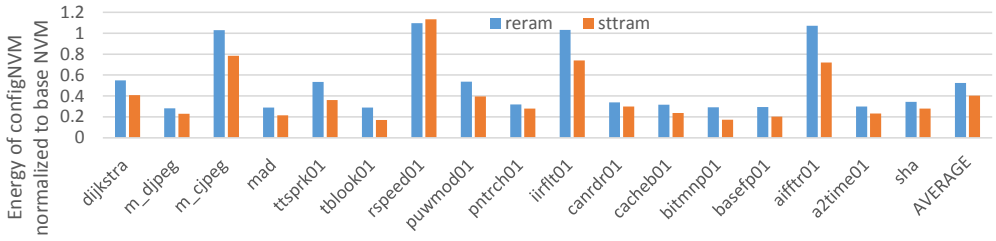

Figure 4: Energy consumption of configNVM normalized to the base NVM

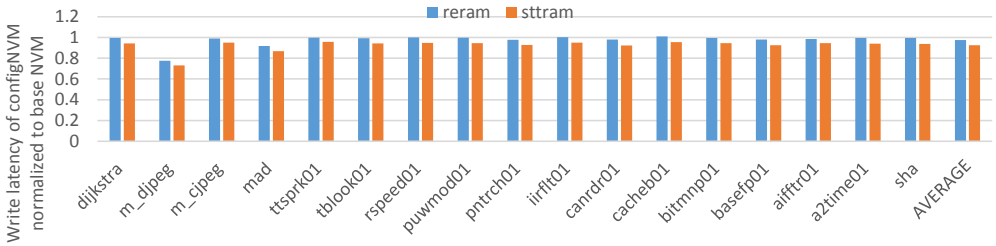

Figure 5: Write latency of configNVM normalized to the base NVM

4.2 Energy Savings

Figure 4 depicts the ReRAM and STTRAM configNVMs' energy consumption normalized to the base NVM cache for a single execution of each of the benchmarks. To consider a worst case scenario, we used a variation of our heuristic (Section 3) in which the configuration did not revert to the base configuration when the heuristic could not find a better configuration than the base. Rather than reverting to the base configuration, the heuristic used the best configuration other than the base, i.e., the best among all the other explored configurations.

On average over all the benchmarks, compared to the base ReRAM cache, ReRAM configNVM cache achieved 48% energy savings, with savings ranging from 45% for *dijkstra* to 72% for *m_djpeg*. The energy was reduced mainly because of the significant reduction in leakage power when the cache banks were shut down as determined by our tuning heuristic. In addition, the dynamic energy was also reduced when the set associativities were adapted to the access requirements of the executing applications. However, for four out of the fifteen benchmarks (*m_cjpeg, rspeed01, iirflt01*, and *aifftr01*), the ReRAM configNVM increased the energy by an average of 6%, with an increase of up to 10% for *rspeed01*. This energy degradation can be prevented (or amortized over multiple executions) by reverting to the base configuration for applications for which configNVM degraded the energy, as is the case with our original heuristic.

On the other hand, compared to the base cache, the STTRAM configNVM achieved 60% energy savings on average over all the benchmarks, with savings ranging from 22% for *m_cjpeg* to 82% for *bitmnp01*. Unlike in the case of ReRAM, the STTRAM configNVM increased the energy for only *rspeed01* by 13%. We observed some significant differences in the configNVM's impact on the energy consumption when using ReRAM versus STTRAM. For example, even though ReRAM configNVM degraded *aifftr01's* and *iirflt01's* energy consumption by 7% and 4%, respectively, STTRAM improved the energy consumption for both benchmarks by 28% and 26%, respectively.

In general, STTRAM configNVM outperformed ReRAM configNVM in energy savings for all the benchmarks, except for *rspeed01*, for which STTRAM's energy increase was higher than that of ReRAM. These results illustrate the optimization potential achievable by configNVM, even with a naive tuning heuristic. We expect the energy savings to be even higher with a more robust and intelligent tuning heuristic, which we intend to explore in future work. The results also show that while configNVM achieves significant energy savings on average, the choice of the specific NVM technology used for implementing the configNVM may differ for different applications and use-cases. Other design tradeoffs that must be considered include design cost, endurance, scalability, etc. For example, even though STTRAM outperforms ReRAM in energy savings, STTRAM is also known to be more expensive and less scalable than ReRAM [5]. Thus, use-case-specific design tradeoffs and goals must be carefully considered when deciding which technologies to use for implementing configNVMs.

4.3 Write Latency

Our goal in exploring configNVM was to optimize the cache's energy consumption without significant latency degradation. However, we also found that configNVM nominally improved the write latency as compared to the base NVM. Figure 5 depicts the ReRAM and STTRAM configNVMs' write latency normalized to the base NVM. On average over all the benchmarks, ReRAM configNVM reduced the write

latency by 2%, and by up to 23% for *m_djpeg*. ReRAM configNVM degraded *cacheb01's* latency by 1%, but the latency for all the other applications either improved or remained the same. Similarly, STTRAM configNVM reduced the average write latency by 7.5%, and by up to 27% for *m_djpeg*. STTRAM configNVM did not degrade any benchmark's write latency.

4.4 Tuning Overheads

We measured the tuning overheads imposed by configNVM with respect to an application's tuning performance and energy overheads. We calculated the tuning performance overhead as the number of tuning stall cycles incurred while determining an application's best configuration, given as *(number of configurations explored - 1) * tuning stall cycles* [4, 16]. We calculated the tuning energy overhead as the energy consumed during the tuning process, when inferior configurations are executed while determining an application's best configuration. Table 1 depicts the benchmarks, number of configurations explored, number of tuning intervals in a single execution, and number of tuning stall cycles. The number of tuning stall cycles on average over all the applications was 3176 cycles, which imposed a maximum of 0.02% performance overhead relative to the tuning interval of 10 ms (20 million cycles). Since the tuning stall cycles were so few relative to the tuning interval, the tuning energy overheads were also infinitesimal and imposed minimal dent on the overall energy savings. These overheads were negligible because of our relatively large tuning interval, and are likely to increase with a smaller tuning interval or a higher tuning frequency, such as in phase-based tuning [3], where configurations are adapted to application phase changes rather than application changes.

Furthermore, we also observed that the tuning overheads imposed by configNVM amortized rapidly during application execution iterations. Figure 6 depicts the number of iterations required to determine the best configurations for each application. On average over all the application, two iterations were required to determine the best configuration, with as many as 3.3 iterations required for *mad*, and as few as 0.7 iterations for *m_cjpeg*. Some applications required more than one iteration to determine the best configuration because there were more configurations explored by our heuristic than tuning intervals in a single execution of the applications (Table 1). The number of iterations required can be reduced even further by using a tuning heuristic/algorithm that explores fewer configurations per application.

Table 1: Benchmarks, number of configurations explored, number of tuning intervals, and tuning stall cycles

Benchmark	Configs explored	# of intervals	stall cycles
dijkstra	14	8	3458
m_djpeg	13	5	3192
c_djpeg	10	14	2394
mad	13	4	3192
ttsprk01	11	6	2660
tblook01	13	5	3192
rspeed01	11	4	2660
puwmod01	12	6	2926
pntrch01	11	8	2660
iirflt01	12	4	2926
canrdr01	15	6	3724
cacheb01	13	5	3192
bitmnp01	13	17	3192
basefp01	18	12	4522
aifftr01	11	10	2660
a2time01	16	9	3990
sha	14	9	3458

5. CONCLUSIONS

In this paper, we proposed configNVMs as a viable optimization for non-volatile memory-based caches in low-power embedded systems, in order to reduce the energy consumption. ConfigNVMs significantly reduce the energy consumption by adapting NVM-based caches' configurations to varying application execution requirements, without imposing significant performance overheads. We discussed potential approaches for implementing configNVMs and empirically showed that configNVMs reduce the average energy consumption by up to 60% as compared to an NVM cache with fixed configurations. We also showed that different NVM technologies perform differently for different applications, thus careful analysis must be performed to determine the appropriate technology for specific use-cases.

For future work, we intend to develop additional robust and low-overhead cache tuning heuristic/algorithms that take into account the different NVM technologies' characteristics and application behaviors. We also intend to explore the benefits of phase-based tuning for configNVMs where configurations are adapted to different application phases to achieve finer-grained optimization. Finally, we intend to explore other novel techniques complementary to configNVMs for minimizing the write latency and energy consumption.

6. REFERENCES

[1] Arm. http://www.arm.com. Accessed: January 2016.

[2] The embedded microprocessor benchmark consortium. http://www.eembc.org/. Accessed: January 2016.

[3] T. Adegbija, A. Gordon-Ross, and A. Munir. Phase distance mapping: a phase-based cache tuning methodology for embedded systems. *Design Automation for Embedded Systems*, 18(3-4):251–278, 2014.

[4] T. Adegbija, A. Gordon-Ross, and M. Rawlins. Analysis of cache tuner architectural layouts for multicore embedded systems. In *Performance Computing and Communications Conference (IPCCC), 2014 IEEE International*, pages 1–8. IEEE, 2014.

[5] R. Aitken, V. Chandra, J. Myers, B. Sandhu, L. Shifren, and G. Yeric. Device and technology implications of the internet of things. In *VLSI Technology (VLSI-Technology): Digest of Technical Papers, 2014 Symposium on*, pages 1–4. IEEE, 2014.

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. *Computer Architecture News*, 40(2):1, 2012.

[7] D. Compiler. Synopsys inc, 2000.

[8] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen. Circuit and microarchitecture evaluation of 3d stacking magnetic ram (mram) as a universal memory replacement. In *Design Automation Conference, 2008.*

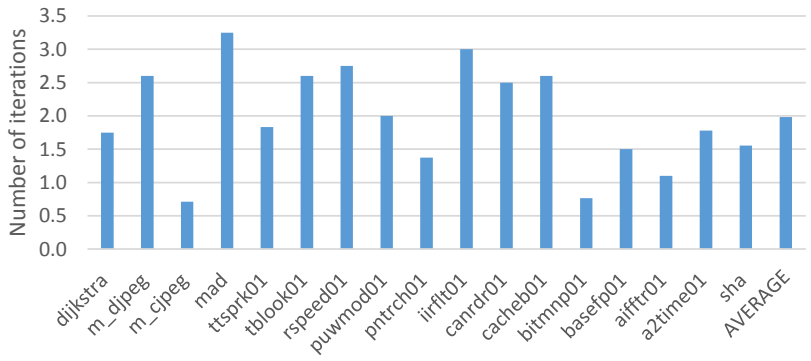

Figure 6: Number of iterations required to determine the best configuration

DAC 2008. 45th ACM/IEEE, pages 554–559. IEEE, 2008.

[9] X. Dong, C. Xu, Y. Xie, and N. P. 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*, 31(7):994–1007, 2012.

[10] A. Gordon-Ross and F. Vahid. A self-tuning configurable cache. In *Proceedings of the 44th annual Design Automation Conference*, pages 234–237. ACM, 2007.

[11] A. Gordon-Ross, F. Vahid, and N. D. Dutt. Fast configurable-cache tuning with a unified second-level cache. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(1):80–91, 2009.

[12] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14. IEEE, 2001.

[13] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das. Cache revive: architecting volatile stt-ram caches for enhanced performance in cmps. In *Proceedings of the 49th Annual Design Automation Conference*, pages 243–252. ACM, 2012.

[14] O. Mutlu and L. Subramanian. Research problems and opportunities in memory systems. *Supercomputing Frontiers and Innovations*, 1(3), 2014.

[15] M. Rasquinha, D. Choudhary, S. Chatterjee, S. Mukhopadhyay, and S. Yalamanchili. An energy efficient cache design using spin torque transfer (stt) ram. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, pages 389–394. ACM, 2010.

[16] M. Rawlins and A. Gordon-Ross. An application classification guided cache tuning heuristic for multi-core architectures. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 23–28. IEEE, 2012.

[17] S. Sharifi, A. K. Coskun, and T. S. Rosing. Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor socs. In *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, pages 873–878. IEEE Press, 2010.

[18] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan. Relaxing non-volatility for fast and energy-efficient stt-ram caches. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 50–61. IEEE, 2011.

[19] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen. A novel architecture of the 3d stacked mram l2 cache for cmps. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 239–249. IEEE, 2009.

[20] X. Wu, J. Li, L. Zhang, E. Speight, and Y. Xie. Power and performance of read-write aware hybrid caches with non-volatile memories. In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, pages 737–742. IEEE, 2009.

[21] C. Xu, X. Dong, N. P. Jouppi, and Y. Xie. Design implications of memristor-based rram cross-point structures. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[22] C. Xu, D. Niu, X. Zhu, S. H. Kang, M. Nowak, and Y. Xie. Device-architecture co-optimization of stt-ram based memory for low power embedded systems. In *Proceedings of the International Conference on Computer-Aided Design*, pages 463–470. IEEE Press, 2011.

[23] W. Xu, H. Sun, X. Wang, Y. Chen, and T. Zhang. Design of last-level on-chip cache using spin-torque transfer ram (stt ram). *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(3):483–493, 2011.

[24] C. Zhang, F. Vahid, and W. Najjar. A highly configurable cache for low energy embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(2):363–387, 2005.

[25] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. Energy reduction for stt-ram using early write termination. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 264–268. ACM, 2009.