FTSPM: A Fault-Tolerant ScratchPad Memory

Amir Mahdi Hosseini Monazzah[1], Hamed Farbeh[2], Seyed Ghassem Miremadi[3], Mahdi Fazeli[4], and Hossein Asadi[5]
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran 11155-9517
Email: [1]ahosseini@ce.sharif.edu, [2]farbeh@mehr.sharif.edu, [3]miremadi@sharif.edu, [4]m_fazeli@ce.sharif.edu, and [5]asadi@sharif.edu

***Abstract*—ScratchPad Memory (SPM) is an important part of most modern embedded processors. The use of embedded processors in safety-critical applications implies including fault tolerance in the design of SPM. This paper proposes a method, called FTSPM, which integrates a multi-priority mapping algorithm with a hybrid SPM structure. The proposed structure divides SPM into three parts: 1) a part is equipped with *Non-Volatile Memory* (NVM) which is immune against soft errors, 2) a part is equipped with Error-Correcting Code, and 3) a part is equipped with parity. The proposed mapping algorithm is responsible to distribute the program blocks among the above three parts with regards to their vulnerability level. The simulation results demonstrate that the FTSPM reduces the SPM vulnerability by about 7x in comparison to a pure SRAM-based SPM. In addition, the dynamic energy consumption of the proposed method is 77% and 47% less than that of a pure NVM-based SPM and a pure SRAM-based SPM, respectively.**

Keywords*—*Reliability, Mapping of SPM, SPM, Non-Volatile Memory.

I. Introduction

Energy consumption, performance, and reliability are the major concerns in designing embedded processors [1]. These concerns are mainly affected by on-chip memory cells that constitute about 60% of the chip area [2]. On-chip memory cells, i.e., cache and scratchpad memories (SPMs), have been widely used to decrease the energy consumption and to improve the performance. A comparison between cache and SPM shows that SPM requires less area and energy than cache memory because of the absence of tag array and controller circuits in SPM [3]. In addition, the use of embedded processors in real-time applications are developed explosively [4]. Here, predictability is one of the major requirements of the real-time applications. In comparison to software-managed SPM, hardware-controlled cache memory complicates the predictability of the system [3]. Based on this fact and due to lower power consumption of SPM, cache has been replaced by SPM in many embedded processors [5].

One of the main applications of embedded processors is in *Safety-Critical Real-Time* systems, where the reliability of SPM is of decisive importance. Soft errors due to radiation-induced bit-flips are a major contributor affecting the reliability of SPMs. With continuous down scaling of emerging technology and the vulnerability paradigm shift from *Single Event Upsets* (SEUs) to *Multiple-Bit Upsets* (MBUs), SPMs have become more vulnerable to soft errors [6].

Almost all previous studies dealing with radiation-induced soft errors in SPMs are based on either duplicating the memory contents, or using traditional memory protection methods, e.g., *Error Correction Codes* (ECC) [2], [7], [8]. Duplicating the memory contents imposes high overheads in terms of power and die size while ECCs have severe limitations on correcting MBUs.

SPM mapping algorithms that deal with allocating SPM space to program blocks are one of the major challenges in exploiting SPM. Since various program blocks have different vulnerability to soft errors, mapping algorithms have direct effects on the reliability of SPM; however, the reliability of SPM has not been considered in previous studies during the mapping phase.

This paper proposes a Fault-Tolerant method for SPM, called FTSPM, which integrates a multi-priority reliability-aware mapping algorithm within a hybrid fault-tolerant SPM structure. The proposed hybrid structure supports three levels of protection: 1) a *Non-Volatile Memory* (NVM) which is immune against soft errors, 2) a SRAM part protected with ECC, and 3) a parity-protected SRAM. The proposed mapping algorithm is responsible to distribute the program blocks among the above three parts with regards to their vulnerability level. Using NVM cells in the SPM structure results in the following advantages:

1) The different structure of NVM cells from the traditional SRAM cells completely immunes some of these memory technologies against radiation-induced soft errors [9]. Consequently, we can immunize parts of SPM area without imposing any protection redundancy overhead to these parts.
2) Since NVMs have ultra-low leakage power [10], using these memory cells alongside of SPM space significantly decreases SPM static energy consumption.

Due to some limitations of NVMs, e.g., write latency and endurance (maximum number of write operations that an NVM cell can tolerate), SRAMs should be used in conjunction with NVMs to take advantages of low latency and high endurance of SRAMs and low leakage power of NVMs [10]. To strike a balance among *Reliability*, *Performance*, *Power*, and *NVM Endurance*, the proposed multi-priority mapping algorithm allocates different SPM areas to different program blocks according to program blocks vulnerabilities. The proposed algorithm is also able to optimize the mapping of program blocks for reliability, performance, power, or endurance according to

system requirements.

The remaining of this paper is organized as follows. In Section II, previous work is reviewed. In Section III, the proposed hybrid SPM structure and mapping algorithm are explained. An example which helps to understand the details of FTSPM is presented in Section IV. Section V describes the simulation setup and results. Conclusions are presented in Section VI.

II. Related Work

Most of the previous studies in SPM have management focused on proposing an optimized mapping algorithm to minimize energy consumption or to maximize the system performance. The SPM mapping algorithm is responsible to manage the limited SPM space and map the most frequently accessed blocks of application to SPM space. SPM management is an optimization problem that can be handled by the programmer or the compiler [11]. Basically, there are two approaches to map program blocks to SPM: *static approach* and *dynamic approach*. In the static approach, a subset of program blocks are transferred to SPM when the application starts and there is no block transfer between the off-chip memory and SPM during the application execution. In the dynamic approach, program blocks can be transferred between SPM and the off-chip memory during the application execution [12].

In addition to performance and energy consumption, SPM as an SRAM-based on-chip memory plays a major role in the reliability of embedded systems. It is a well-known fact that SRAM cells are extremely susceptible to radiation-induced errors, i.e. soft errors [13]. As technology shrinks toward nanometer era, they become even more vulnerable to these errors [14].

Many investigations have been done to protect cache and main memory against MBUs; however, there are a few studies that focused on protecting SPMs against even SEUs. Thus in this section, the previous studies including the methods for improving the performance or energy consumption of SPM are introduced first and then the previous work on SPM reliability is investigated.

A. Improving Performance and/or Energy Consumption of SPM

In [15], a dynamic mapping algorithm, which maps the code section of the programs to SPM, has been introduced. In this study, coarse- and fine-grained program blocks are considered. In the coarse-grained mode, program blocks are constructed from functions, and in the fine-grained mode, a sequence of instructions constructs a block. After partitioning the program to the blocks, the number of accesses to each block is computed by a static profiling. Based on the profiling result, the most frequent accessed blocks are selected as candidates for mapping to the SPM. Then, the SPMs energy consumption and performance of each mapping scenario are calculated and the most efficient mapping scenario is selected for implementation. The method which used in this study has also been exploited in many researches on SPM mapping algorithms.

A dynamic mapping algorithm has been developed for code section of the programs in [16] which implies hardware modifications to the system. This study has introduced a SPM controller unit that records the corresponding mapping address of each block on the SPM space. The main difference of this study and [15] is in the implementation of dynamic transferring of program blocks to the SPM. This study has suggested adding a new type of commands to the *Istruction Set Architecture* (ISA) of processor, named as *SPM Mapping Instruction (SMI)*. SMI commands that stall the processor are executed before the execution of candidate blocks. After this interrupt, the candidate block is copied from its current address in the off-chip memory to the allocated SPM space, which registered in the SPM controller unit. Then in the execution of program is resumed.

In [17], a dynamic SPM allocation algorithm for mapping data section of programs has been presented. This study has concentrated on mapping of arrays to SPM space. Unlike the studies in [15] and [16], in this work, the related commands for mapping the arrays are also generated automatically. The proposed algorithm consists of three consequent steps. First, the SPM space is partitioned into the sections with different sizes. Each section of the SPM is then divided into different parts with the same size alongside the section. In the second step, the total amount of execution time related to each array as well as the reference aggregation to the specific part of each array is determined based on static profiling. Finally, the candidate parts of arrays are selected and the transition commands which will be added to the code are generated automatically.

In recent years, the trend of using NVMs (e.g., STT-RAM, MRAM, and PCM) in the design of caches and SPMs has been increased; however, all of the previous work proposing to use NVM-based SPMs have only concentrated on improving performance or energy consumption of the system.

A dynamic SPM allocation algorithm has been proposed in [10] to transfer the best subset of application blocks between off-chip memory and hybrid SRAM-NVM SPM. Due to high dynamic energy consumption and write latency of NVM, write-intensive data blocks are mapped to the SRAM part and read-intensive data blocks are mapped to the NVM part. In this way, the write endurance of NVM and the energy consumption and the latency of the SPM are improved.

In [18], the energy overhead and the latency of write operations into NVM cells are significantly improved by decreasing the retention time (the time which an NVM cell could correctly sustain its value) in the STT-RAM cells . The proposed algorithm has tried to allocate the STT-RAM part to blocks with least life-time while keeping other blocks in the SRAM part across the SPM space.

B. Improving SPM Reliability

To the best of our knowledge, among previous work for improving SPM reliability, only three methods have targeted dealing with soft errors caused by high-energy particle strike while other studies have considered the reliability of SPM against thermal fluctuation across the SPM space. In this section, first the previous methods to cope with thermal fluctuation are reviewed and then studies on protecting the SPM against soft errors are introduced.

In [19], a dynamic compiler-based mapping algorithm has been proposed which concentrates on mapping data parts of program to SPM. In the first step of the algorithm, the most frequently accessed blocks of program are determined. These

blocks mostly include the program loops. In the next step, the candidate loops are partitioned based on their iterations. Finally, the algorithm decides to map those loops only in some of their iterations and leaves the remainder of loop iteration for the cache. In this way, a thermal balance is formed between cache and SPM and the algorithm prevents the formation of hot-spots alongside of SPM and cache space.

Unlike the method presented in [19] which is only applicable on the systems utilizing cache and SPM simultaneously, a method has been introduced in [20] to improve the thermal reliability of SPM without engaging other parts of the system. This study has introduced an algorithm based on the *regularity* or *irregularity* of memory access sequence in each program. For regular access patterns, a hardware unit manipulates the address bits of accessed blocks in order to distribute the blocks alongside the SPM space and prevents the reference aggregation at a specific part of SPM. For those programs that have irregular access patterns, the program codes are analyzed and program blocks are categorized into two groups, namely *hot variables* and *cold variables*. Then, the mapping algorithm tries to map some cold variables between each hot variable in order to form a thermal balance along SPM space.

As mentioned earlier, there are few studies concentrating on the reliability of SPM against radiation-induced soft errors. Indeed, these studies have focused on protecting the SPM blocks against soft errors without considering the vulnerability of blocks that should be mapped to the SPM space. The proposed method in [3] is based on data block duplication under the control of compiler. This method does not guarantee to duplicate all data blocks and provides no solution for updating the replicas.

In [8], with interpretation of *Redundant Array of Independent Disks* (RAID) systems for memories, distributed SPMs in multicore systems are protected against soft errors. To reduce energy consumption of extra SPMs accesses due to RAID architecture, an aggressive voltage scaling is applied to the system which leads to exponentially increase in the vulnerability of SPMs against soft errors.

In [7], the reliability of instruction part of SPM has been increased by the means of traditional protection techniques. The SPM space in this study is assumed to remain unchanged during the program execution. Because of this assumption, the proposed method cannot be applied to data part of SPM due to frequent data update. In addition, the reliability achievement of this method is limited to the ability of applied detection and protection techniques to deal with bit flips, which decreases with technology scaling.

III. FTSPM: Proposed SPM Structure and Mapping Algorithm

In this section, the hybrid SPM structure as well as the mapping algorithm of *Fault-Tolerant SPM (FTSPM)* is explained in detail. To the best of our knowledge, this is the first work that has proposed to use NVM along SRAM cells to design a reliable and low-power SPM structure. In addition, this work has introduced a reliability-aware hybrid SPM space mapping algorithm. For the NVM part of SPM, STT-RAM technology is exploited which is the most promising NVM technology for on-chip memories [21].

In [9], the reliability of STT-RAM against high-energy particle strike and thermal fluctuation have been evaluated. It has been reported that unlike SRAM cells, STT-RAM cells are completely immune against particle strikes. Furthermore, considering the results, the probability of errors caused by thermal fluctuation would be less than 10^{-15} for a year, which is significantly less than the probability of SRAM radiation-induced transient errors. These results confirm that STT-RAM cells would be considerably more reliable than SRAM cells against radiation-induced transient errors as well as thermal fluctuations.

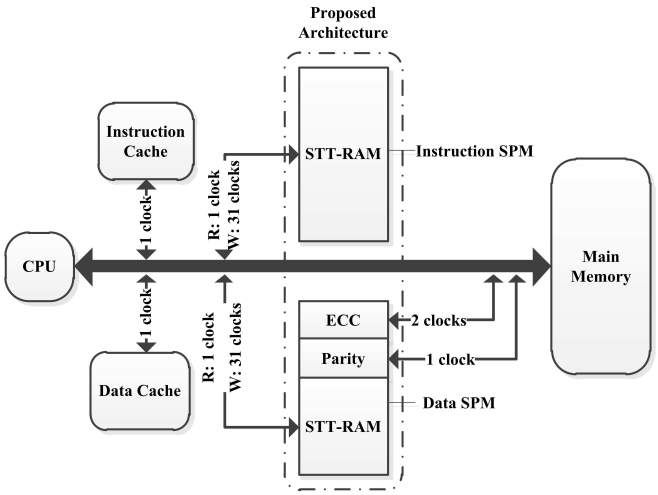

Fig. 1. The interaction of the proposed architecture in a system

Due to significantly higher robustness of STT-RAM cells against soft errors as compared to SRAM cells, FTSPM proposes to partition SPM area to a STT-RAM section and a SRAM section; and to map the program blocks to the SPM hybrid space such that the reliability is enhanced while providing an efficient trade-off between performance, energy consumption, and STT-RAM endurance. To provide a trade off between reliability, performance, and energy consumption, SRAM section is also partitioned to a parity protected part and a *Single Error Corrected-Double Error Detected* (SEC-DED) protected part. The proposed approach is based on the following key observations:

- Vulnerabilities of various program blocks to soft errors are not the same. In other words, the probability of a faulty block to produce an erroneous output is different for various program blocks.
- Fully STT-RAM-based SPM structure satisfies the reliability requirement; however it endures performance and energy consumption overheads of write operations in addition to limited STT-RAM endurance.
- Overhead of protecting fully SRAM-based SPM against soft errors is significantly higher than partially SRAM-based SPM.
- Software controlled SPM allows to manage available SPM space according to the required level of reliability.

According to these observations, a hybrid SPM structure is proposed to improve the reliability of SPM while taking advan-

Algorithm 1 Mapping Determiner Algorithm (MDA)

Input: Data and Code Blocks of Program
Output: Proper Position of each Block across the hybrid SPM

while any block exist **do**
  if (current block == code block) and (current block size $\leq$ instruction SPM size) **then**
    map the current block to instruction SPM
  end if
  if (current block == data block) and (current block size $\leq$ STT-RAM size in data SPM) **then**
    map the current block to STT-RAM part of data SPM
  end if
end while
while any block exist in STT-RAM part of data SPM **do**
  current block susceptibility $\leftarrow$ number of block's reference * it's life time
  constructing a descending order susceptibility list of blocks in STT-RAM section of data SPM
end while
{Checking the performance overhead of current mapping scenario}
while performance overhead of current mapping scenario $>$ performance threshold **do**
  omit the least susceptible block from STT-RAM of data SPM
  update susceptibility list
  calculate performance overhead of current mapping scenario
end while
{Checking the power overhead of current mapping scenario}
while power overhead of current mapping scenario $>$ power threshold **do**
  omit the least susceptible block from STT-RAM of data SPM
  update susceptibility list
  calculate power overhead of current mapping scenario
end while
{Checking the endurance of current mapping scenario}
while any block exist in STT-RAM part of data SPM **do**
  if number of write in current block $>$ write cycles threshold **then**
    omit current block from STT-RAM part of data SPM
  end if
end while
{Determining the position of evicted blocks from STT-RAM part of data SPM}
$avg_{sus} \leftarrow calculate$ "avg. susceptibility over evicted blocks"
while any block exist in STT-RAM part of data SPM **do**
  if (current block susceptibility $\geq avg_{sus}$) and (current block size $\leq$ ECC size in data SPM) **then**
    Map the current block to ECC part of data SPM
  end if
  if (current block susceptibility $\leq avg_{sus}$) and (current block size $\leq$ Parity size in data SPM) **then**
    Map the current block to Parity part of data SPM
  end if
end while

tages of both SRAM and STT-RAM technologies to overcome their limitations. Fig. 1 shows the suggested structure for SPM.

The main challenge for the proposed hybrid structure is to distribute program blocks between the STT-RAM section and the SRAM section such that the system requirements are satisfied. As mentioned, STT-RAM cells are immune against soft errors and their static power is significantly lower than that of SRAM cells; however, they suffer from limited endurance, high latency, and high dynamic power of write operation. On the other hand, the limitations of STT-RAMs are not experienced in SRAM cells but they have their own drawbacks. Firstly, in nano-scale technologies (45nm and beyond), the static power is becoming the dominant factor of the total power consumption [22]. This limits the use of SRAM cells for on-chip memories in nano-scale technologies. In addition, it is a well-known fact that SRAM cells are highly vulnerable to radiation-induced errors especially in [23] where particle strikes may cause MBUs. ECCs can be used to protect SRAM cells against soft errors; however, the area and power consumption of ECCs significantly increases when designed for detection and correction of multiple bit flips.

According to the above discussions, the proposed SPM structure consists of three regions with different characteristics in term of reliability, performance, power, and endurance. The proposed mapping algorithm is also responsible to generate a reliable SPM allocation without noticeably affecting other parameters. From the reliability point of view, all of the program blocks are better to be mapped to the STT-RAM region; from the performance and dynamic energy points of view, all the program blocks are better to be mapped to the parity-protected SRAM region and finally, it is not efficient to map write intensive blocks to the STT-RAM region for endurance point of view. The proposed mapping algorithm considers these extreme points and tries to allocate the more reliable SPM regions to more vulnerable program blocks without violating performance, energy, and endurance budget.

The mapping algorithm consists of two phases; an off-line phase and an on-line phase. The off-line phase which is named *Mapping Determiner Algorithm* (MDA) is responsible for determining each program block to be mapped to which SPM region. Algorithm 1 represents this phase of the algorithm. The inputs of this off-line phase are the profiling information of the application. The second phase is responsible for on-line transferring of blocks between SPM and the off-chip memory.

Prior to applying Algorithm 1, a pre-characterization of program blocks is accomplished based on the profiling information to distinguish which blocks should be mapped to SPM. Afterward, Algorithm 1 specifies the SPM region that should be allocated to each block in the following six steps:

1) All data blocks and instruction blocks are mapped to the STT-RAM region of *D-SPM* and *I-SPM*, respectively. We have proposed to use fully STT-RAM I-SPM instead of hybrid structure because the write operation overhead and STT-RAM endurance is not a concern for read-only instruction blocks. Instruction mapping is accomplished in this step and the algorithm continues in the next five steps to deallocate a subset of data blocks from the STT-RAM region and allocate the SEC-DED protected and the parity protected SRAM region to them. This

TABLE I. RESULTS OF PROFILING CASE STUDY PROGRAM

Block Name	Number of Reads	Number of Writes	Average Number of Reads in each Reference	Average Number of Writes in each Reference	Number of Stack Calls	Maximum Stack Size Needed (Byte)	Life-Time (Cycles)
Main	3,327,700	0	2,620	0	397,561	348	2,086,576
Mul	25,973,000	0	40,710	0	6,400	72	4,221,439
Add	906,200	0	1,433	0	7,100	72	193,356
Array1	2,181,630	1,114,894	10,800	5,519	0	0	4,217,662
Array2	1,113,200	484	5,538	2	0	0	4,215,929
Array3	2,178,000	1,113,684	10,835	5,540	0	0	4,207,400
Array4	1,113,200	484	5,538	2	0	0	4,205,142
Stack	234,009	177,052	1	1	0	0	19,813

is performed in order to satisfy performance, energy consumption, and endurance budgets.

2) Data blocks mapped to STT-RAM region are sorted according to their vulnerability to soft errors. Vulnerability of a block is calculated as the multiplication of *the number of block reference* and its *life-time*.
3) The performance overhead of the current SPM allocation scenario is calculated. If the performance overhead exceeds its predefined threshold (a custom predefined percentage of overhead from the ideal situation), a data block with the least vulnerability is removed from STT-RAM region and then this step will be re-executed. The process is then repeated until satisfying the performance constraint.
4) The energy overhead of the current SPM allocation scenario is calculated. If the energy overhead exceeds its predefined threshold, a data block with the least vulnerability is removed from the STT-RAM region. This step will be re-executed until satisfying the energy requirement.
5) To satisfy the endurance of the STT-RAM region, the number of writes to each STT-RAM-allocated data block is calculated and all the blocks with write-cycles greater than STT-RAM write threshold are removed from the STT-RAM region, regardless of their vulnerability.
6) After satisfying performance, energy, and endurance thresholds in the previous steps, the blocks that have been removed from the STT-RAM region would be assigned to the SEC-DED protected or parity protected region of SRAM. This is done based on blocks vulnerability and size limitations.

After specifying the SPM region of each data block, the sequence of blocks accesses will be extracted from the static profiling information. Based on this sequence, the exact SPM address of each block and the sequence of blocks transfer, i.e., the exact point of mapping and un-mapping of blocks during application execution will be generated. In the next step, instructions that transfer program blocks between off-chip memory and SPM are inserted in proper lines of the code to transfer the blocks at run-time.

IV. MOTIVATIONAL EXAMPLE

In this section, the efficiency of the FTSPM mapping algorithm and the corresponding mapping tool is verified by considering a case study example. For this purpose, the pseudo code presented in Algorithm 2 is executed on the simulation platform. This program contains two multiply functions, two add functions, and a quick sort library function using four arrays as their inputs. The size of each array is about 2 KB.

The SPM configuration of the platform used in this experiment consists of a 16 KB instruction SPM and a 16 KB data SPM. Data SPM constitutes of a 2 KB ECC protected SRAM region, a 2 KB parity protected SRAM region, and a 12 KB STT-RAM region which is completely immune against radiation-induced soft errors; the instruction SPM is completely implemented by STT-RAM cells.

After executing the considered program on the simulation platform, the information shown in Table I is retrieved by means of application profiling. As it is shown, the programs are separated into 8 different blocks, which consists of data blocks and instruction blocks. Based on this profiling, the information which is necessary for the second phase is achieved.

After completing the profiling phase, the MDA algorithm is called. Based on the information extracted from profiling phase, in this algorithm, the proper place of each block in the proposed hybrid structure is determined. Among all of the program blocks in Table I, the *Main* block could not be mapped to the instruction SPM because of the size limitation in the 16 KB instruction SPM. The *Add* and the *Mul* blocks will be mapped to instruction SPM since their sizes are small enough to be mapped to the instruction SPM and no writing operation is done in these blocks. So they do not limit the endurance of STT-RAM cells. It should be noted that the primary write operations which are done during coping of these blocks from the main memory to SPM, have not been considered in the

Algorithm 2 Case study program

Input: Array1, Array2, Array3 and Array4
Output: Addition and multiplication of arrays and sorting the Array1

1: Initializing Array1, Array2, Array3 and Array4
2: $i \leftarrow 0$
3: **while** i<100 **do**
4: $Array1 \leftarrow Multiply(Array1, Array2)$
5: $Array3 \leftarrow Multiply(Array3, Array4)$
6: $Array1 \leftarrow Add(Array1, Array2)$
7: $Array3 \leftarrow Add(Array3, Array4)$
8: **end while**
{Sorting the Array1}
9: $Array1 \leftarrow Qsort(Array1)$
{Qsort is a library function and it uses stack frequently.}

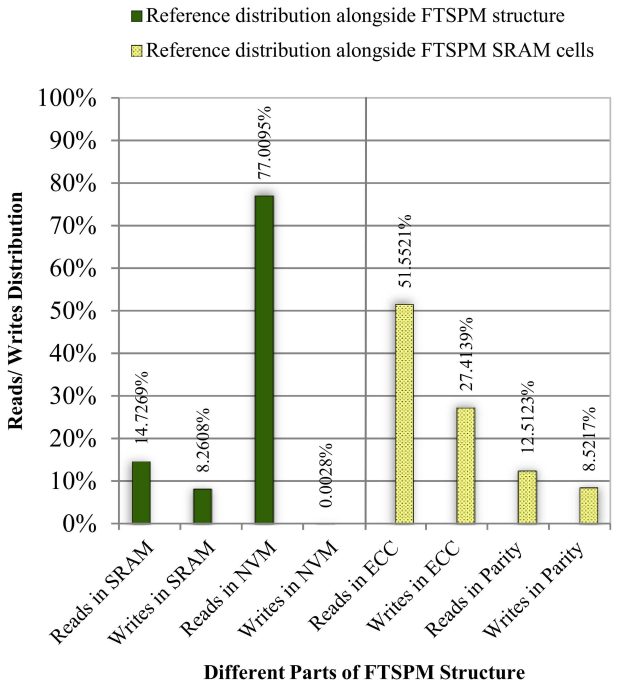

Fig. 2. Distribution of read/write operations across the FTSPM structure

Table I, as these operations are performed just once before the first running of the blocks .

The blocks of *Array1*, *Array3*, and *Stack* are removed from the STT-RAM part of data SPM, because of their intensive write operation which violates the write threshold on the STT-RAM region. *Array3* and *Array4* blocks can be mapped to the STT-RAM region of data SPM. Among the blocks which have been removed from the STT-RAM region, the *Stack* block is mapped to the parity part of data SPM based on its vulnerability to radiation-induced soft errors. *Array1* and *Array3* blocks are also mapped to the ECC region of data SPM.

The developed profiler tool used in this experiment also reports the number of stack calls during each reference to an instruction block and its required stack size during that reference. This helps the MDA Algorithm to map instruction blocks with their desirable stack area, whenever those instruction blocks are mapped to the instruction SPM. Another important factor reported after profiling is the life-time of the blocks. The lifetime of a block is the total duration of time periods across the program execution, which is started by referring that block and ended by the first reference to the other blocks by the program counter. The output of the MDA Algorithm has been shown in Table II.

After mapping the blocks to the SPMs, mapping and unmapping commands are set and located in the proper position within the main source code, based on the sequence of program execution achieved by static profiling. In addition, the address of each block in the SPM is determined in this step. These processes are done by an automatic tool which is developed as a part of this work.

After determining the positions of all blocks across the SPM and modifying the source code to implement the proper mapping scenario, the application is ready to be executed on the proposed SPM structure. For verifying the effectiveness of

TABLE II. MAPPING DETERMINER ALGORITHM OUTPUT FOR CASE STUDY PROGRAM

Block Name	Mapping of SPM	STT-RAM/SRAM
Main	No	–
Mul	Yes	STT-RAM
Add	Yes	STT-RAM
Array1	Yes	SRAM(ECC)
Array2	Yes	STT-RAM
Array3	Yes	SRAM(ECC)
Array4	Yes	STT-RAM
Stack	Yes	Parity

the proposed method, the execution of the new code is also profiled.

The primary information to validate the results of mapping scenario is the manner of blocks distribution across the hybrid structure. In Fig. 2, the distribution of read and write operations for the case study program has been shown. The reported percentages for the ECC and parity regions have been calculated based on the total read and write operations occurring alongside the SRAM cells.

Indeed, the hybrid structure affects the primary properties of the SPM, e.g., *Reliability*, *Performance*, *Energy Consumption*, and *Endurance*. In the following, we explain how the reliability of the proposed method has been calculated.

Based on different vulnerabilities among the regions of the hybrid SPM structure, the equation used for calculating the reliability should be aware of two fundamental parameters. The first one is the percentage of references to each region of the hybrid SPM or distribution pattern of the program blocks across the SPM; the second parameter is the vulnerability of each region against radiation-induced soft errors.

Errors in a system can be categorized in the following three types [6]:

- *Silent Data Corruption (SDC)*: In this error type, the appearance of the errors is not detected in the target system.
- *Detectable Un-recoverable Error (DUE)*: This category refers to the errors detected by the protection techniques, but the corrupted data cannot be recovered.
- *Detectable Recoverable Error (DRE)*: This category refers to the errors that can be detected and recovered by the protection techniques.

The conventional parity protection technique can detect single bit error and the conventional ECC, i.e. SEC-DED, is capable of detecting two bits error or correcting single bit error. Thus, the major challenge in determining the reliability of the proposed method is to calculate the probability distribution of the one or multi-bit errors caused by particle strikes.

The rate of bit-flips in different technology node has been reported in [6]. According to this study, if it is assumed that a radiation-induced soft error has occurred alongside the 40-nm technology size, the probabilities of one, two, three, and more than three bit-flips are about 62%, 25%, 6%, and 7%, respectively.

Based on the above information and the *Architectural Vulnerability Factor* (AVF) [24], the reliability of the proposed

method is computed by considering the following formulas:

$$Vulnerability = SDC_{AVF} + DUE_{AVF} \quad (1)$$

$$\begin{aligned} SDC_{AVF} = & \sum_{i=0}^{n} (ACE_{time} of ParityBlock_i \\ & \times SDC_{probability} of ParityBlock_i) \\ & + \sum_{i=0}^{m} (ACE_{time} of ECCBlock_i \\ & \times SDC_{probability} of ECCblock_i) \end{aligned} \quad (2)$$

$$\begin{aligned} DUE_{AVF} = & \sum_{i=0}^{n} (ACE_{time} of ParityBlock_i \\ & \times DUE_{probability} of ParityBlock_i) \\ & + \sum_{i=0}^{m} (ACE_{time} of ECCBlock_i \\ & \times DUE_{probability} of ECCblock_i) \end{aligned} \quad (3)$$

$$DUE_{probability} inParity = P(1\ bit\ Corruption) \quad (4)$$

$$DUE_{probability} inECC = P(2\ bits\ Corruption) \quad (5)$$

$$SDC_{probability} inParity = P(\geq 2\ bits\ Corruption) \quad (6)$$

$$SDC_{probability} inECC = P(\geq 3\ bits\ Corruption) \quad (7)$$

The *Architecturally Correct Execution* (ACE) Time used in the above equations is the percentage of execution time in which the block is vulnerable to the fault.

After considering the distribution of read/write operations and using the calculated formulas, the reliability of the case study program which was executed on the FTSPM structure is about 86% while the reliability of the corresponding execution on the baseline ECC-protected SRAM-based SPM was about 62%. In addition, since the amount of writes which had done on the STT-RAM region of SPM is efficiently controlled by the MDA Algorithm, the performance degradation is negligible. Furthermore, for the same reason the dynamic energy consumption is 44% lower than the baseline SRAM SPM; and as it was expected, the static energy consumption is significantly lower than the baseline SRAM SPM (56% reduction was observed).

As mentioned, the write endurance of STT-RAM cells is one of the major challenges in using this memory technology in the SPM. Table III shows the endurance of SPM for a pure STT-RAM SPM and FTSPM. As reported, the proposed FTSPM structure and the mapping algorithm significantly increase the endurance of the SPM. Since there is no common idea about the threshold number of writes that a STT-RAM cell could tolerates, the thresholds between lower and upper bounds which can be found in the articles [2] were considered in Table III.

Considering the performance of the system, using the NVM technologies in on-chip memories may increase the execution time because of its extra write cycle duration in comparison to SRAM-based memories. To overcome this NVMs drawback, the FTSPM algorithm considers this challenge through the primary stage of mapping and deports the write intensive blocks from the STT-RAM region of the SPM. Furthermore, it can be seen in Fig. 1 that the read latency of STT-RAM is only one clock cycle; while for the ECC-protected SRAM region

TABLE III. COMPARISON OF ENDURANCE BETWEEN BASELINE PURE STT-RAM SPM AND PROPOSED STRUCTURE

Number of Writes Threshold	Baseline Pure STT-RAM SPM	FTSPM
10^{12}	∼40 Minutes	∼ 61 Days
10^{13}	∼7 Hours	∼1.5 Years
10^{14}	∼3 Days	∼16 Years
10^{15}	∼28 Days	∼166 Years
10^{16}	∼3 Months	∼1665 Years

of the SPM, the read and write latencies are two clock cycles. Thus, the total amount of time savings on read operations, and penalties on write operations with considering the mapping strategies lead to almost the same performance on the FTSPM and the baseline SRAM SPM.

V. SIMULATION SETUP AND RESULTS

To evaluate the proposed approach, *FaCSim*, a cycle-accurate ARM processor simulator is used [25]. In the experiments, FTSPM has been compared to two baselines SPM structures, i.e., a pure SRAM-based structure protected by SEC-DED and a pure STT-RAM-based structure. A pure STT-RAM-based structure is completely immune against radiation-induced soft errors. The first baseline suffers from high static power and vulnerability to MBUs and the second one suffers from the endurance as well as energy consumption and the latency of write operations. The detail characteristics of each structure are presented in Table IV.

The latency and the energy consumption of the memory subsystem are calculated using *NVSIM* [26]. *Synopsis Design Compiler*© [27] is also used to measure the latency and energy consumption of the parity and SEC-DED combinational circuits. *MiBench* benchmark suite [28] has been used as the workload. Performance, energy consumption, endurance, and reliability of the system which runs this benchmark suite are measured to evaluate the efficiency of the proposed method.

Dynamic energy consumption per access of each region is depicted in Fig. 3, while the static power consumption of the proposed method, baseline SRAM, and baseline STT-RAM are 7.1 mW, 15.8 mW, and 3 mW, respectively. As mentioned in section III, the reliability of the SPM is measured based on *Architectural Vulnerability Factor* (AVF) [24]. To measure

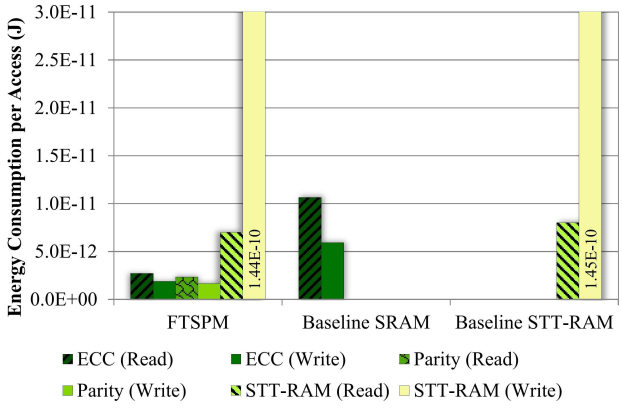

Fig. 3. Dynamic energy consumption per access in different structures

TABLE IV. CONFIGURATION PARAMETERS USED IN $FaCSim$

	Baseline pure SRAM SPM				Baseline pure NVM SPM				FTSPM			
	Type	Size	Read Latency	Write Latency	Type	Size	Read Latency	Write Latency	Type	Size	Read Latency	Write Latency
Cache Inst./Data	(1)	8KB	1 Clock	1 Clock	(1)	8KB	1 Clock	1 Clock	(1)	8KB	1 Clock	1 Clock
Inst. SPM	(3)	16KB	2 Clocks	2 Clocks	(4)	16KB	1 Clock	10 Clocks	(4)	16KB	1 Clock	10 Clocks
Data SPM	(3)	16KB	2 Clocks	2 Clocks	(4)	16KB	1 Clock	10 Clocks	(2)	2KB	1 Clock	1 Clock
									(3)	2KB	2 Clocks	2 Clocks
									(4)	12KB	1 Clock	10 Clocks

(1): Unprotected SRAM (2): Parity protected SRAM (3): SEC-DED protected SRAM (4): STT-RAM

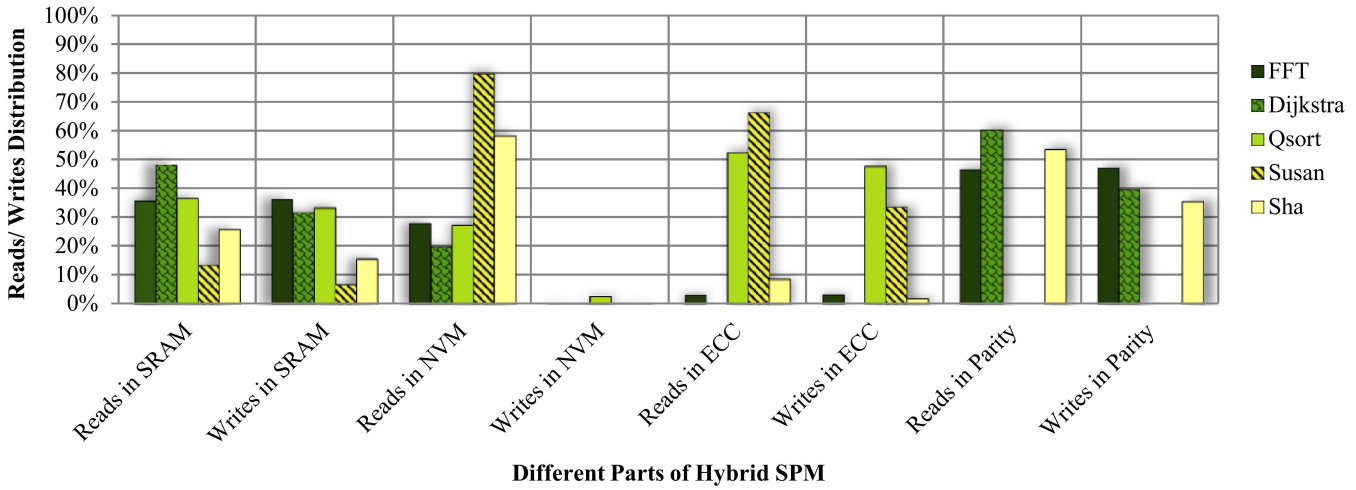

Fig. 4. Distribution of read/write operations alongside FTSPM structure

the vulnerability of the SPM, vulnerable intervals of each block is multiplied by the probability of MBUs in the case of particles strike, which is reported in [6]. Fig. 4 illustrates the read/write distribution of each benchmark alongside FTSPM structure. Fig. 5 presents the vulnerability of FTSPM and the pure SRAM SPM. It is noteworthy that the pure STT-RAM SPM is supposed to be immune against soft errors.

According to Fig. 5, the vulnerability of the pure SRAM SPM is about 7x more than FTSPM. As it can be observed in Fig. 5, the vulnerability of the baseline SRAM structure is a constant value and it is independent from the behavior of the workload. It was revealed that this observation relies on the distribution of radiation induced soft errors across the surface of uniform baseline SRAM structure and non-uniform FTSPM structure. As it was anticipated, the shortcoming of ECC method against MBUs resulted in higher vulnerability of the baseline SRAM structure, while the robustness of the NVM part of FTSPM structure against MBUs and the proper distribution of more vulnerable blocks across more reliable area in FTSPM resulted in less SPM vulnerability.

Static energy consumption of the baseline structures and FTSPM is depicted in Fig. 6. As expected, the static energy consumption of FTSPM is significantly less than that of the pure SRAM SPM due to replacement of a large fraction of SRAM cells by STT-RAM cells; however, the static energy consumption of FTSPM is higher than that of pure STT-RAM SPM due to high static energy of included SRAM cells. Static energy consumption of the proposed hybrid SPM and pure

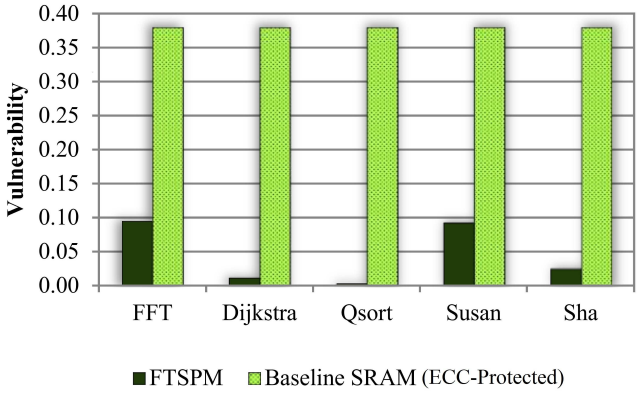

Fig. 5. Vulnerability results for different structures

STT-RAM SPM is about 45% and 25% less than that of the of the pure SRAM SPM, respectively.

On the other hand, as presented in Fig. 7, the dynamic energy consumption of FTSPM is 47% less than that of the pure SRAM SPM and 77% less than that of pure STT-RAM SPM. Besides, lower read energy and higher write energy of STT-RAM cells compared to SRAM cells and the intelligent distribution of program blocks in SPM regions using the FTSPM mapping algorithm made the FTSPM hybrid structure considerably more dynamic energy efficient than the pure SPM and the pure STT-RAM SPM baseline.

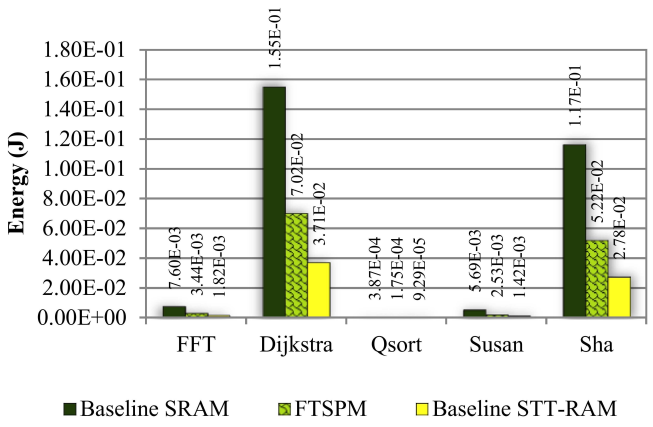

Fig. 6. Static energy consumption results for different structures

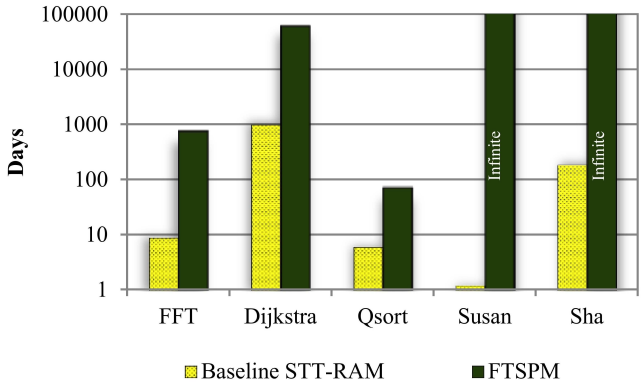

Fig. 8. Endurance results for different structures

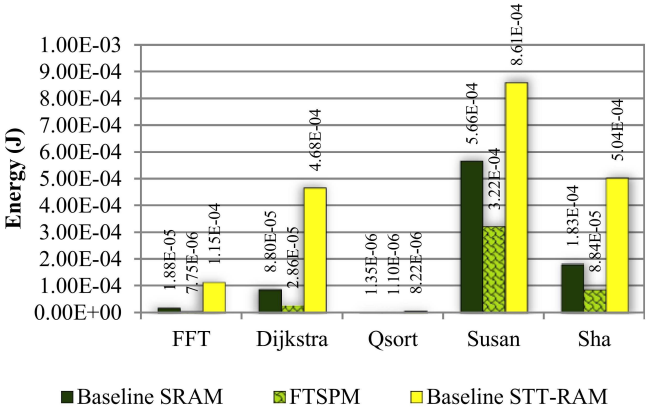

Fig. 7. Dynamic energy consumption results for different structures

Fig. 8 illustrates the endurance of FTSPM and the pure STT-RAM SPM baseline. Fig. 8 confirms that by distributing program blocks between STT-RAM and SRAM regions of SPM, the proposed algorithm enhances the STT-RAM endurance of the hybrid SPM compared to a pure STT-RAM SPM by three orders of magnitude. Note that the endurance of the pure SRAM SPM is not reported because it is supposed that there is no endurance limitation in the SRAM cells. Finally, due to the strategy of the FTSPM algorithm which tries to decrease the write aggregation of program alongside STT-RAM cells, the simulation results shows that the performance overhead of the proposed method is negligible in comparison to pure SRAM-based SPM.

VI. Conclusion

This paper proposed a method which called FTSPM to protect SPM against soft errors. FTSPM utilizes a hybrid STT-RAM/SRAM structure for SPM in order to improve the reliability of SPM. FTSPM also employs a reliability-aware mapping algorithm to allocate SPM hybrid regions to program blocks. According to the simulation results, vulnerability of the FTSPM structure to soft error is about 7x less than that of the pure SRAM-based SPM baseline, in addition to about 55% and 47% reduction in static energy and dynamic energy consumption, respectively. Furthermore compared to the pure STT-RAM-based SPM baseline, FTSPM increases the endurance of SPM by three orders of magnitude and decreases the dynamic energy consumption up to 23%. Moreover, the performance overhead of FTSPM is less than 1%.

References

[1] P. Marwedel, Embedded systems design, Second edition, Springer, 2010.

[2] International Technology Road-map for Semiconductors (ITRS), “ERD_ ERM 2010 final report memory assessment,” Final report, 2010.

[3] F. Li, G. Chen, M. Kandimer, “Improving scratch-pad memory reliability through compiler-guided data block duplication,” Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD 05), pp. 1002-1005, USA, November 6-10, 2005.

[4] I. Lee, J. Leung, S. Son, Handbook of real-time and embedded systems, First edition, Chapman and Hall/CRC Computer and Information Science Series, 2008.

[5] H. Takase, H. Tomiyama, H. Takada, “Partitioning and allocation of scratch-pad memory for priority-based preemptive multi-task systems,” Proc. Design, Automation and Test in Europe (DATE 10), pp. 1124-1129, Germany, March 8-12, 2010.

[6] A. Dixit, A. Wood, “The impact of new technology on soft error rates,” Proc. IEEE International Reliability Physics Symposium (IRPS 11), pp. 5B.4.1-5B.4.7, USA, April 10-14, 2011.

[7] H. Farbeh, M. Fazeli, F. Khosravi, S. G. Miremadi, “Memory mapped SPM: protecting instruction scratchpad memory in embedded systems against soft errors,” Proc. European Dependable Computing Conference (EDCC 12), pp. 218-226, Romania, May 8-11, 2012.

[8] L. A. D. Bathen, N. D. Dutt, “E-RoC: embedded RAIDs-on-chip for low power distributed dynamically managed reliable memories,” Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE 11) , pp. 1-6, France, March 14-18, 2011.

[9] S. Guangyu, E. Kursun, J. A. Rivers, X. Yuan, “Exploring the vulnerability of CMPs to soft errors with 3D stacked non-volatile memory,” Proc. IEEE International Conference on Computer Design (ICCD 11), pp. 366-372, USA, October 9-12, 2011.

[10] J. Hu, C. J. Xue, Q. Zhuge, W. C. Tseng, E. H. M. Sha, “Towards energy efficient hybrid on-chip scratch pad memory with non-volatile memory,” Proc. Design, Automation and Test in Europe (DATE 11), pp. 1-6, France, March 14-18, 2011.

[11] I. M. Aouad, R. Schott, O. Zendra, “A tabu search heuristic for scratch-pad memory management,” Proc. International Conference on Software Engineering and Technology (ICSET 10), pp. 386-390, Italy, April 28-30, 2010.

[12] I. M. Aouad, O. Zendra, “A survey of scratch-pad memory management techniques for low-power and low-energy,” Proc. International Workshop on Implementation, Compilation, Optimization of Object-

Oriented Languages, Programs and Systems (ICOOOLPS 07), pp. 31-38, Germany, July 30, 2007.

[13] D. Yoon, M. Erez, "memory mapped ECC: low-cost error protection for last level caches," Proc. International Symposium on Computer Architecture (ISCA 09), pp. 116-127, USA, June 24-29, 2009.

[14] D. F. Heidel, P. W. Marshall, J. A. Pellish, K. P. Rodbell, K. A. LaBel, J. R. Schwank, S. E. Rauch, M. C. Hakey, M. D. Berg, C. M. Castaneda, P. E. Dodd, M. R. Friendlich, A. D. Phan, C. M. Seidleck, M. R. Shaneyfelt, M. A. Xapsos, "Single-event upsets and multiple-bit upsets on a 45 nm SOI SRAM," IEEE Transactions on Nuclear Science (TNS 09), vol. 56, no. 6, pp. 3499-3504, December, 2009.

[15] S. Steinke, L. Wehmeyer, B. S. Lee, P. Marwedel, "Assigning program and data objects to scratchpad for energy reduction," Proc. Design Automation and Test in Europe (DATE 02), pp. 409-415, France, March 4-8, 2002.

[16] A. Janapsayta, S. Parameswaran, A. Ignjatovic, "Hardware/software managed scratchpad memory for embedded system," Proc. International Conference on Computer-Aided Design (ICCAD 04), pp. 370-377, USA, November 7-11, 2004.

[17] L. Li, L. Gao, J. Xue, "Memory coloring: a compiler approach for scratchpad memory management," Proc. International Conference on Parallel Architectures and Compilation Techniques (PACT 05), pp. 329-338, USA, September 17-21, 2005.

[18] K. Swaminathan, R. Pisolkar, X. Cong, V. Narayanan, "When to forget: a system-level perspective on STT-RAMs," Proc. Asia and South Pacific Design Automation Conference (ASP-DAC 12), pp. 311-316, Australia, January 30- February 2, 2012.

[19] M. Wang, Y. Wang, D. Liu, Z. Shao, "Improving the reliability of embedded systems with cache and SPM," Proc. IEEE 6th International Conference on Mobile Adhoc and Sensor Systems (MASS 09), pp. 825-830, China, October 12-15, 2009.

[20] M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, H. Corporaal, "Thermal-aware scratchpad memory design and allocation," Proc. IEEE International Conference on Computer Design (ICCD 10), pp. 118-124, Netherlands, October 3-6, 2010.

[21] T. Perez, A. F. Cesar, De-Rose, "Non-volatile memory: emerging technologies and their impacts on Memory Systems," Technical report, Pontificia Universiadae, Brazil, September, 2010.

[22] S. Rodriguez, B. Jacob, "Energy/power breakdown of pipelined nanometer caches (90nm/65nm/45nm/32nm)," Proc. International Symposium on Low Power Electronics and Design (ISLPED 06), pp. 25-30, Germany, October 4, 2006.

[23] V. Chandra, R. Aitken, "Impact of technology and voltage scaling on the soft error susceptibility in nanoscale CMOS," Proc. IEEE International Symposium on Defect and Fault Tolerance of VLSI system (DFT 08), pp. 114-122, USA, October 1-3, 2008.

[24] S. S. Mukherjee, C. T. Weaver, J. Emer, S. K. Reinhardt, T. Austin, "Measuring architectural vulnerability factors," Proc. IEEE Micro, pp. 70-75, USA, December 3-5, 2003.

[25] J. Lee, J. Kim, C. Jang, S. Kim, B. Egger, K. Kim, S. Y. Han, "FaCSim: a fast and cycle-accurate architecture simulator for embedded systems," Proc. ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems, pp. 89-99, USA, June 12-13, 2008.

[26] X. Dong, C. Xu, Y. Xie, N. P. Jouppi, "NVSim: a circuit-level performance, energy, and area model for emerging nonvolatile memory," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD 12), vol. 31, no. 7, pp. 994-1007, July, 2012.

[27] Synopsys Design Compiler, www.synopsys.com, 2010.

[28] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T . Mudge, R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," Proc. International Workshop of the Workload Characterization (WWC 01), pp. 314, USA, December 2, 2001.