Using a Flexible Fault-Tolerant Cache to Improve Reliability for Ultra Low Voltage Operation

ABBAS BANAIYANMOFRAD, University of California, Irvine
HOUMAN HOMAYOUN, George Mason University
NIKIL DUTT, University of California, Irvine

Caches are known to consume a large part of total microprocessor power. Traditionally, voltage scaling has been used to reduce both dynamic and leakage power in caches. However, aggressive voltage reduction causes process-variation–induced failures in cache SRAM arrays, which compromise cache reliability. In this article, we propose FFT-Cache, a flexible fault-tolerant cache that uses a flexible defect map to configure its architecture to achieve significant reduction in energy consumption through aggressive voltage scaling while maintaining high error reliability. FFT-Cache uses a portion of faulty cache blocks as redundancy—using block-level or line-level replication within or between sets—to tolerate other faulty caches lines and blocks. Our configuration algorithm categorizes the cache lines based on degree of conflict between their blocks to reduce the granularity of redundancy replacement. FFT-Cache thereby sacrifices a minimal number of cache lines to avoid impacting performance while tolerating the maximum amount of defects. Our experimental results on a processor executing SPEC2K benchmarks demonstrate that the operational voltage of both L1/L2 caches can be reduced down to 375 mV, which achieves up to 80% reduction in the dynamic power and up to 48% reduction in the leakage power. This comes with only a small performance loss (<%5) and 13% area overhead.

Categories and Subject Descriptors: B.3.2 [**Design Styles**]: Cache Memories; B.3.4 [**Reliability, Testing, and Fault-Tolerance**]: Error-Checking, Redundant Design

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Aggressive voltage scaling, fault-tolerant cache, low voltage operation, remapping

ACM Reference Format:
Abbas Banaiyanmofrad, Houman Homayoun, and Nikil Dutt. 2015. Using a flexible fault-tolerant cache to improve reliability for ultra low voltage operation. ACM Trans. Embedd. Comput. Syst. 14, 2, Article 32 (February 2015), 24 pages.
DOI: http://dx.doi.org/10.1145/2629566

1. INTRODUCTION

On-chip caches are known to consume a large portion (about 30% to 70%) of total processor power [Wong et al. 2007; Zhang et al. 2005], and their size will continue to grow due to device scaling coupled with performance requirements. Therefore, it

This material is based on work supported by the NSF under award CCF-1029783 (Variability Expedition). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.
Authors' addresses: A. BanaiyanMofrad and N. Dutt, Center for Embedded Computer Systems, School of Information and Computer Science, University of California, Irvine, CA 92697; emails: {abanaiya, dutt}@uci.edu; H. Homayoun, Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA 22030; email: hhomayou@gmu.edu.

DOI: http://dx.doi.org/10.1145/2629566

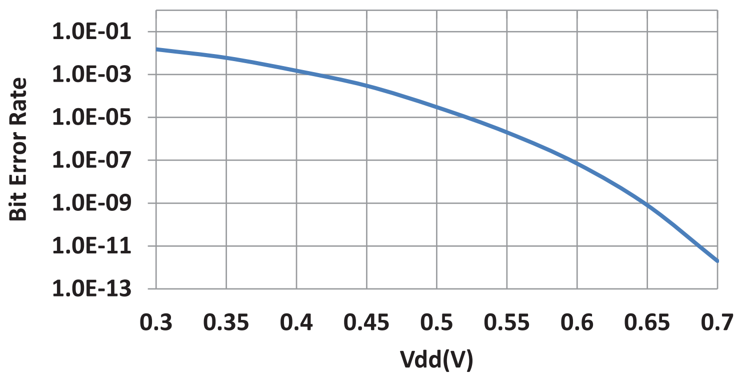


Fig. 1. Probability of SRAM cell bit failure rate in 90nm.

becomes critical to manage the power and reliability of the caches to reduce total power consumption while maintaining the reliability of the entire processor system.

Traditionally, voltage scaling has been used to reduce the dynamic and the leakage power consumption of the cache. However, aggressive voltage scaling causes process-variation–induced failures in the SRAM cells. An SRAM cell can fail due to an access time failure, a destructive read failure, or a write failure [Agarwal et al. 2005; Mukhopadhyay et al. 2005]. Figure 1 represents the failure rate of an SRAM cell based on the operational voltage in a 90 nm technology [Morita et al. 2007; Koh et al. 2009a]. To save power while achieving an acceptable manufacturing yield of 99.9% for 64 KB L1 and 2 MB L2 caches, a minimal operational voltage must be selected. From Figure 1, we can see that the probability of failure for each cache array must be kept at less than 1 out of 1,000 to achieve this yield. Based on this assumption, we estimate that the minimum operational voltage for a 64 KB L1 is 620 mV and for a 2 MB L2 660 mV, and we are not able to further reduce the operational voltage without incurring cell failures.

Since applications may not be tolerant to even a single bit error, caches typically must be operated at a high Vdd to ensure a very low probability of failure, leading to high energy consumption. However, by exploiting mechanisms that allow a cache to become inherently resilient to a large number of cell failures, we can operate the cache at a lower Vdd and thus gain significant energy savings. There is a large body of previous work on fault-tolerant cache design at different layers of abstraction, which we review in Section 5. However, most of them are not efficient for high fault rates arising from operation in the near-threshold regime, and these techniques incur significant overheads. To address these issues, in this work we propose an approach that aims to (1) design a low-power and flexible fault-tolerant cache architecture that can detect and replicate SRAM faults arising from operation in the near-threshold voltage region, (2) minimize nonfunctional or disabled cache area to lessen impact on processor performance; and (3) tolerate cache faults as much as possible when the fault rate is high or voltage is very low.

Considering such design goals, we present the flexible fault-tolerant cache (FFT-Cache), a cache architecture that uses a flexible defect map (FDM) to efficiently tolerate the large number of faults when operating in the near-threshold region. FFT-Cache specifies a portion of faulty cache blocks as redundancy and remaps other faulty cache word lines and blocks to them. This is accomplished by using either block-level or line-level (set-level) replication in the same set or between two or more sets. In the remainder of this article, we refer to every physical cache word line, which may contain multiple cache blocks as a line or set. FFT-Cache leverages its FDM to configure the remapping of faulty data blocks either inside the same set or among different sets. FFT-Cache configuration is done in two phases: first we categorize the cache lines based on *degree of conflict* between their blocks, and then we use a graph coloring algorithm to optimize the faulty block remapping and reduce the granularity of redundancy replacement. Using this approach, FFT-Cache initially attempts to replicate faulty blocks inside each line, and otherwise it attempts to replicate faulty blocks among different lines. Our FFT-Cache approach significantly saves power consumption while incurring only a small (5%) performance degradation and a small (13%) area overhead when operating in near-threshold voltages.

The main contributions of our FFT-Cache approach are summarized as follows:

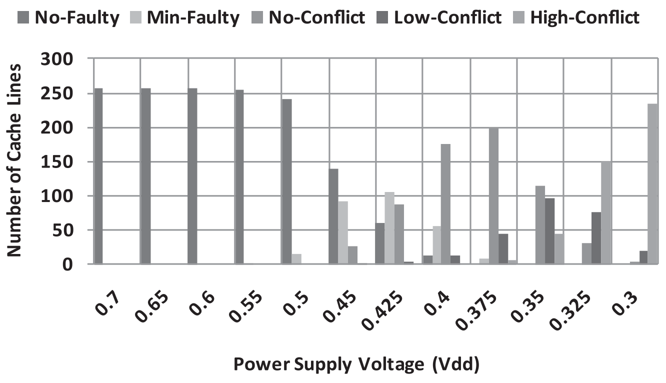
—We categorize the faulty cache lines based on the degree of conflicts among their blocks and then use a portion of them as redundancy to replicate the other ones.
—We deploy a new FDM to simplify the remapping of faulty data blocks both inside a set and among different cache sets with minimal overhead.
—We use a graph coloring algorithm to configure the FDM to optimize the remapping of faulty blocks in the proposed fault-tolerant architecture and minimize the nonfunctional part of the cache.
—We perform a comprehensive design space analysis of power and area overhead of deploying proposed FFT-Cache in L1 and L2 cache hierarchies of an Alpha processor running SPEC benchmarks.
—We compare our scheme with various state-of-the-art cache fault-tolerant schemes and show that our proposed FFT-Cache achieves the lowest operating voltage (375 mV) and the highest power reduction compared to all other techniques.

The rest of this article is organized as follows. Section 2 presents a motivational example of a cache operating in near-threshold voltages. Section 3 introduces the proposed FFT-Cache architecture. Section 4 describes simulation setup and analytical models. Section 5 explores the design space and presents experimental results. Section 6 reviews related work and distinguishes our proposed approach with respect to state-of-the-art cache fault-tolerance techniques. Finally, Section 7 concludes the article.

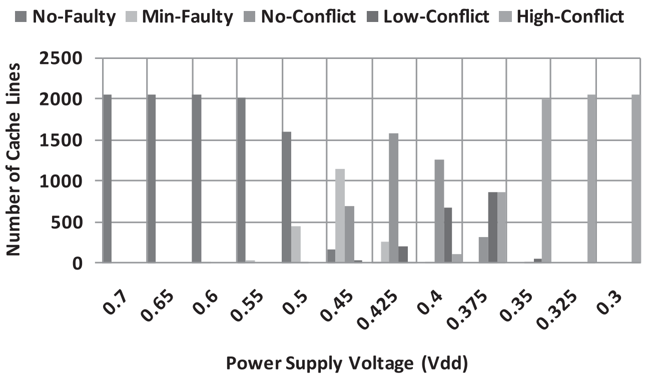
2. MOTIVATION

As can be seen in Figure 1, the failure rate of an SRAM cell increases exponentially when lowering Vdd; consequently, for near threshold voltages, the number of faulty cells is very high, resulting in almost all of the cache lines and blocks becoming faulty. This poses a difficult challenge for the protection of caches while working in the near-threshold voltage regime. To illustrate this concept, we performed a Monte Carlo simulation for both L1 and L2 caches, and the results are presented in Figure 2. Before describing this figure in detail, let us first define some parameters and terminology. For the rest of this article, we refer to every physical cache word line containing a set of blocks as a line or set. The number of blocks in a line or a set equals the associativity of a cache. In addition, each block is divided into multiple equally sized subblocks that can be as small as a single bit or as large as an entire block. Each subblock is labeled faulty if it has at least one faulty bit. Two block/lines have a conflict if they have at least one faulty subblock/block in the same position. As well, we define the max global block (MGB) parameter as the maximum number of blocks in a line that can be set as global (sacrificial) blocks for replication to achieve fault tolerance.

Figure 2 presents the number of caches lines in different categories for a 64 KB four-way set associative L1 and a 2 MB eight-way set associative L2 cache for different Vdd values. In this figure, we categorize the cache lines into five groups:



(a) A 64KB 4-way set associative L1 cache with 64B block size, 8b subblock size, and MGB=1



(b) A 2MB 8-way set associative L2 cache with 128B block size, 8b subblock size, and MGB=2

Fig. 2. Number of cache lines in different categories while varying the supply voltage.

—*No-Faulty*: Include cache lines with no faulty block.
—*Min-Faulty*: Include cache lines with the number of faulty blocks below a certain threshold (i.e., the MGB parameter). We will discuss this parameter in Section 3.2.
—*No-Conflict*: Include cache lines that have multiple faulty blocks but without conflict.
—*Low-Conflict*: Include cache lines with multiple faulty blocks for which their number of conflicts is less than the MGB parameter.
—*High-Conflict*: Include cache lines with multiple faulty blocks and in which the number of conflicts between blocks is more than the MGB parameter.

Figure 2 shows that by increasing the probability of bit failure, the amount of conflicts between blocks in each cache line is increasing. For example, for L1 cache with Vdd values greater than 0.4V, there is no line in the High-Conflict group, but by decreasing the voltage below 0.4V, the amount of High-Conflict lines increases exponentially. Therefore, for caches that operate below 0.4V, it is essential to deal with lines in the High-Conflict group. Previous cache protection techniques consider conflicts between two or more cache lines; however, to the best of our knowledge, none addresses the conflicts within cache lines [Wilkerson et al. 2008; Sasan et al. 2009; Koh et al. 2009a, 2009b], missing the opportunity to more flexibly exploit fault tolerance when operating at very low voltages. In contrast, our FFT-Cache approach is explicitly designed to operate efficiently at very low voltages.

3. PROPOSED ARCHITECTURE

In this section, we first describe the proposed FFT-Cache architecture that uses an FDM to efficiently tolerate SRAM failures. Next, we present the cache configuration that includes FDM generation and configuration stages to configure the architecture for fault-tolerant data access.

3.1. FFT-Cache Organization

FFT-Cache architecture has two banks of data that can be accessed concurrently; they include multiple cache lines, with each line including multiple blocks. FFT-Cache achieves fault tolerance by using a portion of the faulty cache blocks as redundancy to tolerate other faulty cache lines and blocks. Using this approach, FFT-Cache tries to sacrifice a minimal number of cache lines to minimize performance degradation and tolerate the maximum amount of defects at near-threshold operating voltage. This is done by using either block-level or line-level replication in the same set or between two sets. The information of faulty locations is kept in an FDM that is then used to configure the faulty block remapping. To replicate the faulty subblocks in a line (called a *host line*), our scheme tries to find a faulty block in the same line or in another line that has no conflict with other blocks in the host line. We refer to such a block as a Target block. Depending on whether the Target block is in the same line (host line) or in another line, it is called *Local Target Block* or *Global Target Block*, respectively. FFT-Cache always tries to find a local target block. If FFT-Cache cannot find a local target block, it searches for a global target block. Finally, if it fails to find either a local or global target block, it tries to find another faulty line (called a *target line*) that has no conflict with the host line. It then sacrifices the target line to replicate all faulty blocks of the host line. Thus, based on the earlier discussion, the sacrificial target line/block could be one of the following: (1) *Local Target Block*, (2) *Global Target Block*, or (3) *Target Line*.

Note that a local target block can be accessed in the same cycle as the host line and does not require any additional access overhead. This is not true for a global target block or a target line, for which two consecutive accesses are required if the global target block or target line are in the same bank as the host line. To access the host line and global target line/block in parallel, and to minimize the access latency overhead, the host line and target line should be in different banks. In addition, since target blocks/lines do not store any independent data, they are considered nonfunctional. Therefore, the target lines are not addressable as data lines, and thus they are removed from the address scope of the cache. This could impact performance, as it reduces the effective size of the cache. A major advantage of FFT-Cache over other fault-tolerant methods is that we minimize the number of nonfunctional and target lines by initially attempting to find a local target block to sacrifice. If a local target block is not found, FFT-Cache then attempts to find a global target block. Finally, if the first two attempts are not successful, FFT-Cache sacrifices a different cache line as a target line, as do other fault-tolerant methods.

In our fault-tolerant cache architecture, each cache access in low power mode first accesses the FDM. Based on the fault information of the accessed line, the target block/line may be accessed from another bank (in case of a global target block or target line). Then based on the location of target block/line retrieved from the FDM, one or two levels of MUXing are used to compose a fault-free block by choosing appropriate subblocks from both host and target blocks (lines). Figure 3 outlines the flowchart for accesses using FFT-Cache.

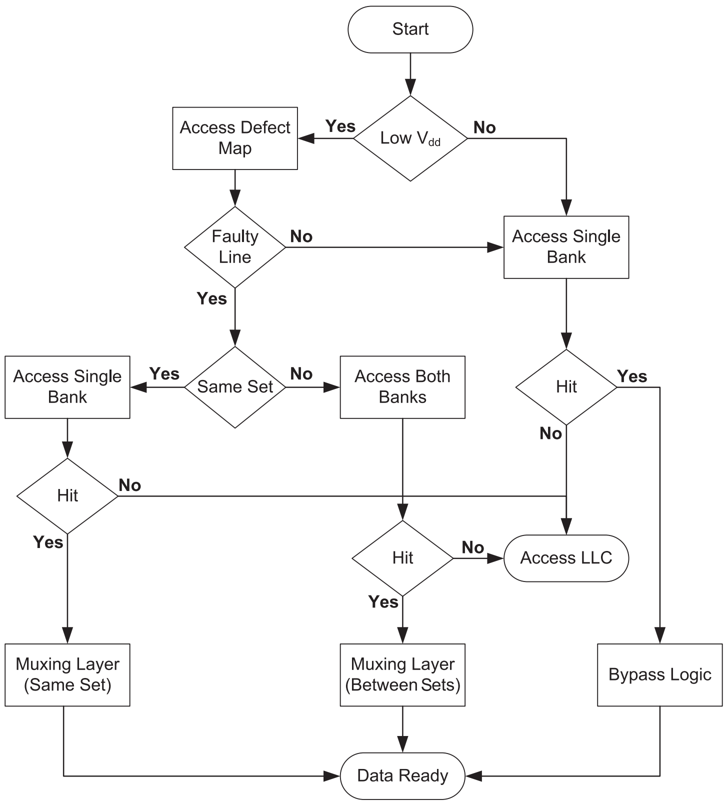


Fig. 3. FFT-Cache access flowchart.

3.2. FFT-Cache Configuration

We now describe the configuration process for FFT-Cache. Initially, a raw defect map is generated at boot time: using the cache memory Built-In Self-Test (BIST) unit, the L1 and L2 cache(s) are tested under low voltage conditions. The output of the BIST is used to initialize the FDM. If there are multiple operating points for different combinations of voltage, temperature, and frequency, the BIST operation is repeated for each of these settings. The obtained defect map is then modified and processed to be usable with FFT-Cache. Updating the FDM is done at full voltage, using a simple algorithm that we explain next. The configuration information can be stored on the hard drive and is written to the FDM at the next system boot-up. In addition, to protect the defect map and the tag arrays, we use the well-studied 8T SRAM cell [Verma and Chandrakasan 2008], which has about 30% area overhead for these relatively small arrays in comparison with 6T SRAM cells. These 8T SRAM cells are able to meet the target voltage in this work for the aforementioned cache structures without failing. An example of an FDM entry for a cache with associativity of 4 is represented in Figure 4.

Each FDM entry includes multiple configuration bits, the defect map of each way (block), the status of each block, and the address of the target line. Each bit in the defect map section represents the faulty state of a subblock. Way Status bits represent the status of each way or block in a cache line. The status of each cache block can assume one of the following: (1) *Nonfaulty*, (2) *Faulty*, (3) *Local Target*, and (4) *Global*. Initially, the status of all blocks in the cache is Nonfaulty, representing the absence of any faulty subblocks. If a block contains at least one faulty subblock, its status will be

WS Bits	Way Status
00	Nonfaulty
01	Local Target
10	Global
11	Faulty

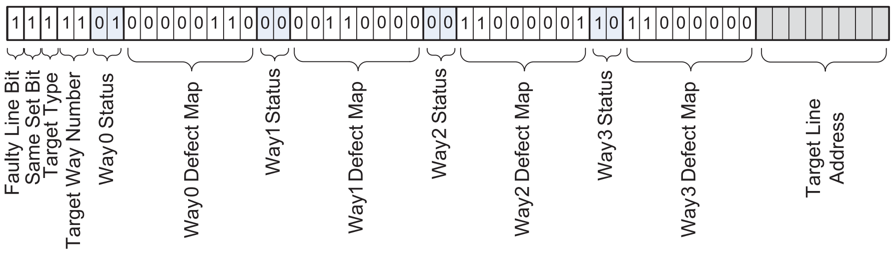


Fig. 4. Details of an entry row of FDM.

Faulty. A block that is used as a target block for other blocks in a line gets the status of Local Target. A block that has a conflict with other blocks in same set cannot be used as a local target block. It gets the status of Global and might be used as a global target block. We define the MGB parameter as the maximum number of blocks in each line that can be sacrificed (set as global blocks) for replication to achieve fault tolerance. For example, in an eight-way set associative cache, if we set MGB $= 4$, then in the worst case we can sacrifice (set as global block) up to four blocks of each faulty line for replication. If we extend it to the whole cache, in the worst case we sacrifice at most half of the cache for fault tolerance.

We present the algorithms for FDM initialization and configuration in Algorithms 1 and 2.

After completion of the FDM initialization algorithm, we run the FDM configuration algorithm.

We use graph coloring to optimize the selection algorithm of both remote target block and remote target line for Step 2 and Step 3 of the FDM configuration algorithm [Klotz et al. 2002]. Graph coloring is an NP-complete problem, and there are many heuristics and transformations available to generate a feasible solution. Accordingly, we use a heuristic graph-coloring algorithm that is a modification of the saturation degree ordering (SDO) algorithm [Al-Omari and Sabri 2006].

We name a graph coloring problem solvable, if for a graph G we can find an integer $K \geq 0$ such that the nodes of G can be colored with K colors while no edge exists between the same colored nodes. We construct a graph based on the conflicts between lines of different entries in the FDM. Each node in this graph represents either a Global Block (*Block Node*) or a faulty entry (representing an entire line) from low- or high-conflict groups (*Line Node*) in the FDM. The edges represent a conflict between a pair of blocks or between a block and a line. For example, two nodes connected by an edge represent two blocks, two lines, or one block and a line that have conflict. Of course, a pair of nodes connected by an edge represents a conflict, and thus it is not possible to set one of these nodes as a remote target block for the other one.

We modify the preceding graph coloring algorithm based on the following constraints:

(1) We force the algorithm to color nodes from at least two different banks.
(2) We force the algorithm to first pick up block nodes and try to color them with line nodes of other banks.

ALGORITHM 1: FDM Initialization Algorithm

```
Input:
  FDM: empty defect map array
Output:
  FDM: initialized defect map array
1: // Step1: Run BIST and find faulty cache lines at a subblock level and fill defect map sections
   of each entry;
2: for i = 1 to |FDM| do
3:   FDM [i].Faulty = False;
4:   FDM [i].SameSet = True;
5:   FDM [i].tLineAddr = i;
6:   for j = 1 to nAssoc do
7:     if FDM [i].block [j].nFSblocks > 0 then
8:       FDM [i].WayStatus [j] = Faulty;
9:       FDM [i].nFblocks ++;
10:     else
11:       FDM[i].WayStatus[j] = NonFaulty;
12:     end if
13:   end for
14:   if FDM[i].nFblocks > 0 then
15:     FDM [i].Faulty = True;
16:   end if
17:   if FDM[i].nFblocks < MGB then
18:     FDM[i].SameSet = False;
19:   end if
20:   if FDM[i].nConflicts > 1 then
21:     FDM[i].SameSet = False;
22:   else
23:     if FDM[i].nConflictingBlocks <= MGB then
24:       set the status of conflicting blocks as Global Block
25:       group other blocks and set one of them as Target Block
26:     else
27:       FDM [i].SameSet = False
28:     end if
29:   end if
30: end for
31: return FDM;
```

Now we apply the modified graph coloring algorithm to our graph to find a solution such that neighboring nodes are not assigned the same color. Therefore, after completion of the coloring algorithm, nodes with the same color are guaranteed to have no edges between them, implying that the corresponding cache lines/blocks have no conflicts between them. We set all nodes with the same color in a group and try to set one of them as a remote target for other nodes in the group. As a result, if a block node has the same color as one or more set nodes in other banks, we set it as their remote target block. If a line node has the same color as one or more line nodes in other banks, we set all of its blocks as a remote target block for the blocks in other lines.

Since the graph coloring algorithm is run during boot time, the algorithm's overhead is amortized within the boot time overhead and is not incurred during runtime. Based on our measurements on a machine with the Intel Core 2 Duo 2 GHz processor and 2 GB memory, the overhead varies from 1ms up to 10ms, depending on cache configuration and fault rate.

ALGORITHM 2: FDM Configuration Algorithm

```
Input:
  FDM: initialized defect map array
Output
  FDM: configured defect map array
 1: // Step1: Traverse the faulty rows of FDM and based on the conflicts between faulty blocks
    inside each row, categorize the FDM entries in different groups;
 2: for i = 1 to |FDM| do
 3:   FDM[i].Faulty = False;
 4:   FDM[i].SameSet = True;
 5:   FDM[i].tLineAddr = i;
 6:   if FDM[i].Faulty then
 7:       if FDM[i].nFblocks < MGB then
 8:           FDM[i].SetStatus = Min_Faulty;
 9:           set the status of faulty blocks to Global Block;
10:       else
11:           if FDM[i].nConflicts == 0 then
12:             FDM[i].SetStatus = No_Conflict;
13:             set the status of one of the faulty blocks to Local Target Block
14:           else if FDM[i].nConflicts == 1 then
15:             FDM[i].SetStatus = Low_Conflict;
16:             make the status of the block that has conflict with other blocks as Global
                Block;
17:           else
18:             FDM[i].SetStatus = High_Conflict;
19:           end if
20:       end if
21:   end if
22: end for
23: // Step2: Assign the lines in Low_Conflict group
24: For the lines in Low_Conflict group, run the graph coloring algorithm to find a Global
    Target Block for each line.
25: // Step3: Assign the lines in High_Conflict group
26: For the lines in High_Conflict group, run the graph coloring algorithm to find a similar line
    from other bank to make it as the Global Target Line.
27: return FDM;
```

Figure 5 shows an example of the FDM configuration for a given distribution of faults in 10 sets of a two-banked four-way set associative cache with four subblocks in each block and MGL = 1[1]. As shown in the figure, Set 1 is clean, without any faulty blocks. Set 2 is a member of the min-faulty group, whereas its faulty block is configured as a remote target block for Set 9. Set 7 is another member of this group, and its single faulty block is set as a global block. Set 3 is an example of a no-conflict group in which the first block (Way1) is set as a local target block to be sacrificed for fault tolerance of the other faulty blocks in the set. Set 6 is another member of this group, with one of its blocks (Way4) set as a local target block. Set 4 is a member of the low-conflict group, in which one of its blocks (Way1) is set as a global block, as it has a conflict with other blocks. The first block of Set 9 is set as a remote target block for this set (Set 4). Set 5 is a member of the high-conflict group, with two conflicts between its blocks (Way0 has conflict with Way2, and Way1 with Way3). All blocks of this set are configured as Remote Target Block for both low-conflict Sets 8 and 10.

[1]Recall that MGL is the maximum number of blocks in a line that can be set as global block.

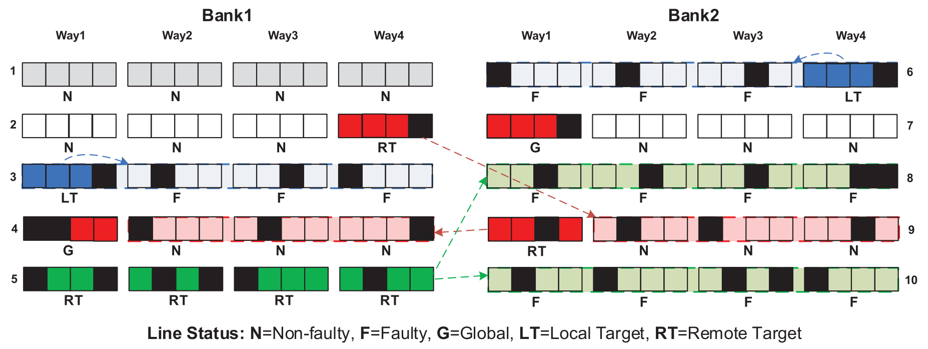
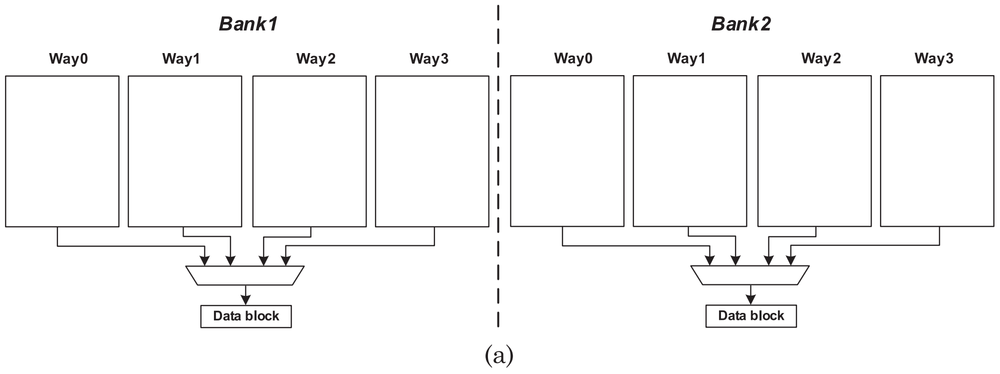

Bank1
Bank2
Way1
Way2
Way3
Way4
N
RT
LT
F
G
Line Status: N=Non-faulty, F=Faulty, G=Global, LT=Local Target, RT=Remote Target

Fig. 5. An example of FDM configuration and remapping for a given distribution of faults in a four-way set associative cache.



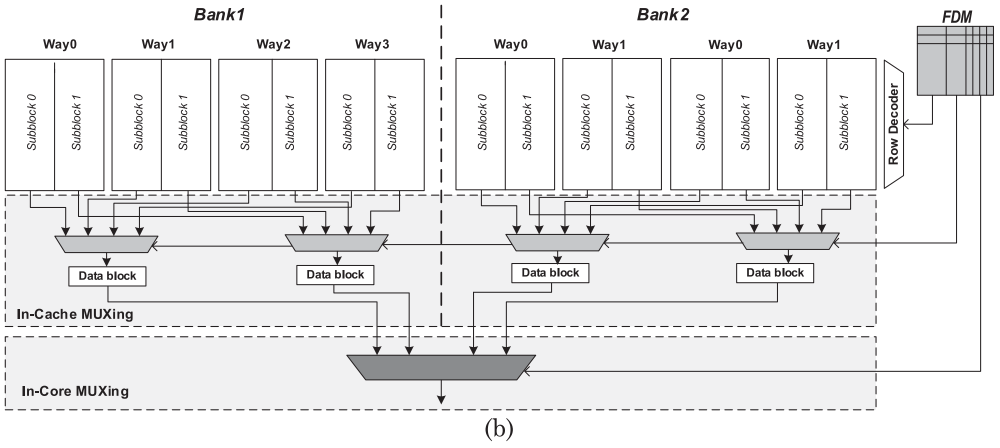


Fig. 6. Architecture details of a conventional four-way set associative cache (a) and the proposed FFT-Cache with FDM and two subblocks per block (b).

3.3. Architecture Details

We now present the architecture of FFT-Cache. We begin with the architecture of a conventional four-way set associative cache with two banks (labeled *CC*) in Figure 6(a). Based on the tag match results, one of the blocks from Way0 to Way3 from either Bank0 or Bank1 is being selected. The data are transferred to and from cache arrays using multiplexers as indicated in the figure.

Figure 6(b) shows the architecture of the proposed FFT-Cache. Let's assume that the cache is divided into two banks, with each bank containing four ways (blocks) that are further divided into two subblocks. The new modules (MUXs and FDM) added to the conventional cache are highlighted in the figure.

The additional multiplexer network would allow the cache to compose the final fault-free line from any of the subblocks in any of the cache ways in each one of cache banks, based on its FDM data. FFT-Cache requires two additional levels of multiplexing to compose the final fault-free line, based on either multiple lines within a set or between two or more sets in different banks. The first multiplexing layer (In-Cache MUXing) is added on the cache side to perform the remapping within a set. Note that this layer replaces the original block-level multiplexer available in set associative caches with multiple smaller subblock-level multiplexers. The second multiplexing layer (In-Core MUXing) is added on the core side to perform remapping between sets in original (local) and target (remote) banks.

The total number of n-to-1 subblock-level multiplexers on the cache side is $k \times b$, where k is the number of subblocks in a block, n is the number of ways (set associativity), and b is the number of banks. For instance, for a two-bank 64KB, four-way set associative cache, with each way with two subblocks, a total of four 4-to-1 multiplexers are required on the cache side. The size of FDM equals the multiplication of number of cache lines by the FDM entry size.

3.4. Hardware Implementation

For a quantitative comparison, we synthesized the MUXing layer and output logic for FFT-Cache as well as the multiplexer and output driver of the conventional cache (CC) using the Synopsys Design Compiler for TSMC 90 nm standard cell library for both L1 and L2 caches. The area and delay of various multiplexers (MUX2, MUX4, ... MUX32) are used to estimate the overall area/delay overhead of the MUXing network in FFT-Cache and the CC in nominal Vdd. We found that the delay of FFT-Cache output logic increased by only 5% compared to CC output MUX network, whereas area and power consumption are increased by only 2% compared to a CC MUX network. Recall that FFT-Cache architecture replaces CC's output MUX with FFT-Cache MUXing layer and output logic; thus, we expect that the proposed mechanism will only result in a minimal increase of the cache delay (estimated at $<5\%$).

4. EVALUATION

This section evaluates the effectiveness of FFT-Cache architecture in reducing power consumption of the processor while keeping overhead as low as possible. Before presenting the experimental results, we describe our methodology and experimental setup, develop an analytical failure model, and outline the exploration space for our experiments.

4.1. Methodology

Table I outlines our experimental setup for the baseline processor configuration. The processor is configured with a 64K four-way set associative L1 cache. The architecture was simulated using an extensively modified version of SimpleScalar 4.0 [Austin et al. 2002] using SPEC2K benchmarks. Benchmarks were compiled with the -O4 flag using the Compaq compiler targeting the Alpha 21264 processor. The benchmarks were fast forwarded for 3 billion instructions, then fully simulated for 4 billion instructions using

Table I. Processor Configuration

L1/Inst cache	2 banks 64 KB, 4 way, 2 cycles,1 port, 64B block size
L2 cache	2 banks 2 MB, 8 way, 20 cycles,1 port, 128B block size
Fetch, dispatch	4 wide
Issue	4 way out of order
Memory	300 cycles
Reorder buffer	96 entry
Instruction queue	32 entry
Register file	128 integer and 125 floating point
Load/store queue	32 entry
Branch predictor	64KB entry g-share
Arithmetic unit	4 integer, 4 floating point units
Complex unit	2 INT, 2 FP multiply/divide units

the reference datasets. We used CACTI6.5 [Muralimanohar et al. 2009] to evaluate area, power, and delay of both L1 and L2 caches and their related FDMs. The Synopsys standard industrial tool chain (with TSMC 90 nm technology library) was used to evaluate the overheads of the MUXing layer components (i.e., MUXes, comparators, MUXes selection logic. etc.).

The load/store latency of two cycles is assumed to be broken into actual cache access taking place in cycle 1, whereas the bus access takes only a part of the next cycle. From our discussion in Section 3.4, the cache delay is increased only slightly ($<5\%$) for nominal Vdd. However, considering the increase in logic delay due to Vdd scaling, we conservatively assume that in the worst case, the latency of the cache would be increased by one full cycle for L1 and two cycles for L2. These extra cycles are considered based on our timing analysis for both MUXing layer (discussed in Section 3.4) and FDM access, which are on the critical path of the cache access.

For a given set of cache parameters (e.g., associativity, subblock size, MGB), a Monte Carlo simulation with 1,000 iterations is performed using our FDM configuration algorithm described in Section 3.3 to identify the portion of the cache that should be disabled while achieving a 99% yield. In other words, probability of failure of the cache must below 0.001 when operating in low power mode.

4.2. Probability of Operation

To estimate the probability of failure for the cache, we developed an analytical model of FFT-Cache that receives the probability of bit failure (p_F) as input. Let's assume an n-way set associative cache with m sets, k subblocks in a block, each of which has d data bits with a fault probability p_F for each bit. We also consider c as the maximum acceptable disabled blocks in a set (MGB) and R as the maximum number of cache sets that can be disabled. We first calculate the probability of cache operation using FFT-Cache scheme. In this case, a cache is operational if all except R sets have at most c disabled blocks in each set. We derive the following equations:

The probability of failure for each subblock that has at least one faulty bit:

$$P_{\text{faulty-subblock}} = P_{\text{fs}} = 1 - (1 - p_{\text{F}})^{d}.$$

The probability of failure for each block that has at least one faulty subblock:

$$P_{\text{faulty-block}} = P_{\text{fb}} = 1 - (1 - p_{\text{F}})^{\text{dk}}.$$

The probability of two blocks being paired with no conflict such that for each pair of subblocks at the same location at least one of them should not be faulty:

$$P_{\text{paired-block}} = P_{\text{pb}} = \left(1 - P_{\text{fs}}^{2}\right)^{\text{k}}.$$

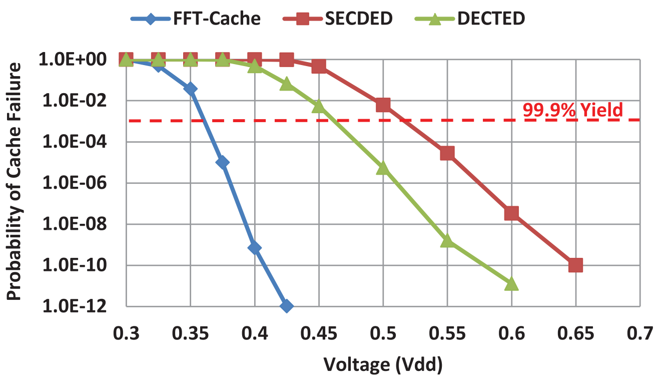


Fig. 7. Probability of cache failure versus Vdd for SECDED, DECTED, and FFT-Cache.

To have an operational set, we need to have a group of at least $n-c$ blocks such that none of them have conflicts with each other. So the probability of finding possible blocks in a set to compose an operational group without any conflict between them such that each pair of them being a paired block without conflict would be

$$P_{\text{group-block}} = P_{\text{gb}} = (P_{\text{pb}})^{\alpha}, \ \alpha = \binom{n-c}{2}.$$

As defined, a set is faulty only if all of its blocks are faulty and none of the possible groups of its blocks is operational. Hence, the probability that a set is functional:

$$P_{\text{set}} = 1 - (P_{\text{fb}})^{n}(1 - P_{\text{gb}})^{\beta}.$$

The probability of a cache set being pairable is that at least one of β possible groups of $n-c$ blocks in it becomes paired to another set after remapping:

$$P_{\text{pairable-set}} = P_{\text{ps}} = \beta(P_{\text{pb}}), \ \beta = \binom{n}{n-c}.$$

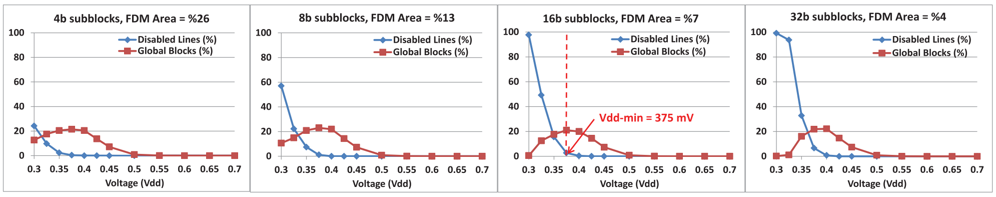
Let's consider R as the maximum number of cache sets that can be nonfunctional but used for replication. Therefore, at most 2R sets could be nonoperational $(1-P_{set})$, from which R sets become paired and operational. The probability that FFT-Cache is operationalis

$$P_{Cache} = \sum_{i=0}^{R} \binom{m}{2i} p_{set}^{m-2i}(1 - P_{set})^{2i} P_{ps}^{i}$$

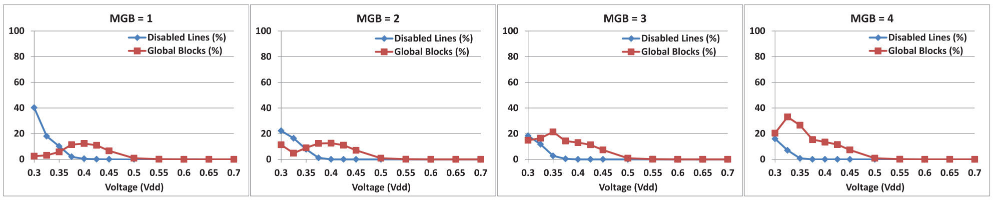
Finally, the probability of cache failure is

$$P_{\text{Cache-failure}} = 1 - P_{\text{Cache}}.$$

We used our analytical models of failure to determine the failure probability of a 64 KB L1 FFT-Cache and compared it to SECDED and DECTED methods with equal area overhead. The results, as shown in Figure 7, demonstrate that at a given voltage, FFT-Cache is the most reliable cache, whereas SECDED is the least reliable cache. If we adopt the definition for Vdd-min as the voltage at which 1 out of every 1,000 cache instances is defective, which equals a 99.9% yield [Wilkerson et al. 2008], based on this figure, FFT-Cache can reduce the Vdd below 375 mV in comparison with 465 mV and 520 mV for DECTED and SECDED methods, respectively.



(a) Percentage of disabled lines and global blocks for four-way L1 (MGB = 1) while varying Vdd and subblock size



(b) Percentage of disabled lines and global blocks for eight-way L1 (4b subblocks) while varying Vdd and MGB

Fig. 8. The effect of changing one design parameter of L1 FFT-Cache while fixing other parameters for different Vdd values.

5. EXPERIMENTAL RESULTS

5.1. Design Space Exploration

We study the impact of various FFT-Cache configuration parameters, including subblock size and MGB, on the number of target lines/blocks (nonfunctional cache part). In addition, we study the impact of cache associativity on FFT-Cache functionality. Note that since MGB is decided by cache associativity (i.e., half of cache blocks in a line can be a global block candidate), it makes a lot of sense to study the impact of cache associativity on the size of nonfunctional cache part.

As mentioned earlier, to evaluate our design for a given set of cache parameters (associativity, MGB, subblock size, and Vdd), a Monte Carlo simulation with 1,000 iterations is performed using the FDM configuration algorithm described in Section 3.2 to identify the global blocks and lines that should be disabled. Our simulation model targets 99% yield for the cache.

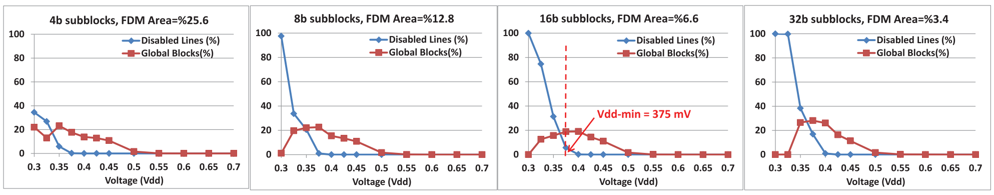
5.2. Vdd-Min Sensitivity Analysis

Figures 8 and 9 present the design space exploration of FFT-Cache for L1 and L2 caches, respectively. We present the results for different associativity (4 and 8) and various subblock sizes (4, 8, 16, and 32) in these figures.

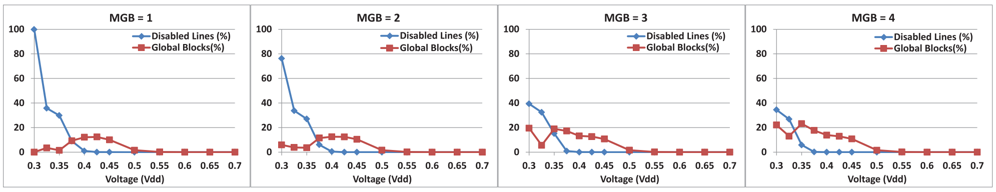
As is evident in the figures, decreasing Vdd increases the size of the nonfunctional part of the cache. It is notable that for very low voltage (below 400 mV), the number of global blocks decreases. Overall, the effective size of cache (cache size – nonfunctional part) decreases as the voltage is lowered. We can also observe from the figure that decreasing the size of subblocks increases the area overhead of the FDM. Decreasing the size of subblocks also reduces the size of the nonfunctional part of the cache.

Increasing MGB increases the number of nonfunctional blocks only slightly while it significantly reduces the number of nonfunctional lines. In fact, increasing the MGB helps the FDM configuration algorithm to find a global target block rather than sacrificing a target line.

During the process of finding FFT-Cache minimum achievable operating voltage (Vdd-min), we limit the size of the nonfunctional part of the cache and the overhead of the FDM table as described next. The number of target lines/blocks (i.e., the size of nonfunctional cache part) determines the performance loss of FFT-Cache. To limit the performance loss, we assume the relative size of the nonfunctional part to be less than

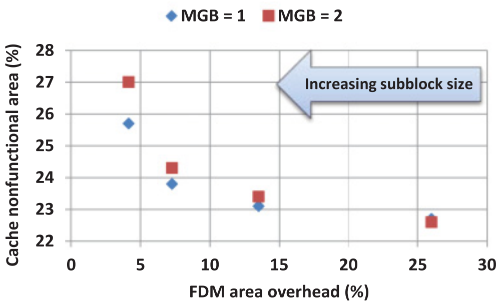


(a) Percentage of disabled lines and global blocks for eight-way L2 (MGB = 4) while varying Vdd and subblock size

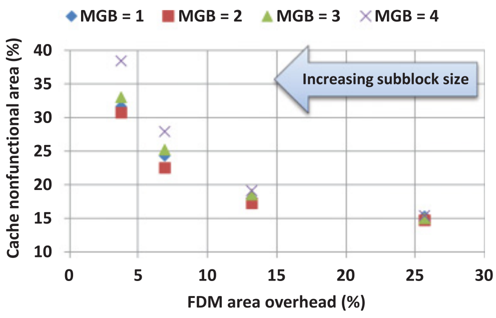


(b) Percentage of disabled lines and global blocks for eight-way L2 (4b subblocks) while varying Vdd and MGB

Fig. 9. The effect of changing one design parameter of L2 FFT-Cache while fixing other parameters for different Vdd values.



(a) Four-way set associative L1



(b) Eight-way set associative L1

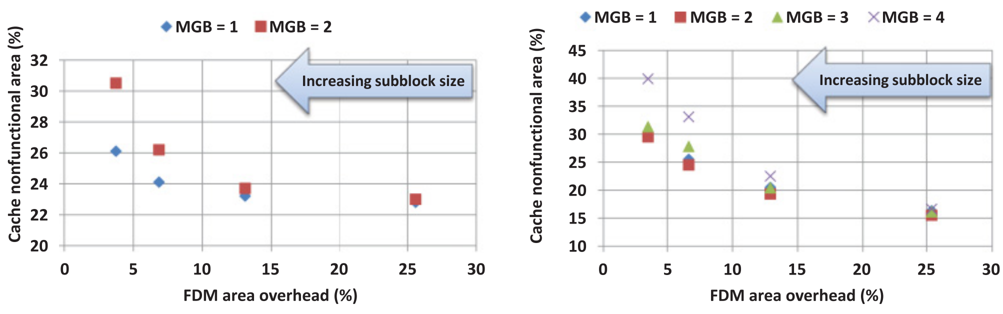
Fig. 10. L1 Design space exploration for different MGB and subblock size pair.

25% of cache size. To minimize the implementation overhead, we assume the FDM size to be less than 10% of cache size. This assumption requires the subblock size to be 16 bits or higher (32 bits and 64 bits).

Based on these assumptions, we find the minimum achievable operating voltage. This is shown in Figures 8 and 9. Using FFT-Cache scheme, the minimum operating voltage for L1 and L2 caches is 375 mV.

5.3. FFT-Cache Parameters Sensitivity Analysis

Figures 10 and 11 present the design space exploration of FFT-Cache with Vdd = 375 mV for L1 and L2 caches, respectively. In these figures, there are two parameters that vary: (1) subblock size (from 4b to 32b) and (2) MGB (from 1 to half of associativity). Here, we try to find a design point that has the minimum sum of both FDM and nonfunctional part overheads. As is evident in these figures, 4b subblock size points have the minimum nonfunctional part overhead, but they have the maximum FDM area overhead for all MGB values. On the other side of this parameter, 32b subblock size points have the minimum FDM area overhead, but the nonfunctional part of such points is more than other points. In Figure 11, if we compare the total overhead between design points of 8b and 16b, the 16b points have less overhead. Thus, for L1 cache, we select a design point with 16b subblock size and MGB = 1 for associativity = 4 and



(a) Four-way set associative L2 (b) Eight-way set associative L2

Fig. 11. L2 design space exploration for different MGB and subblock size pair.

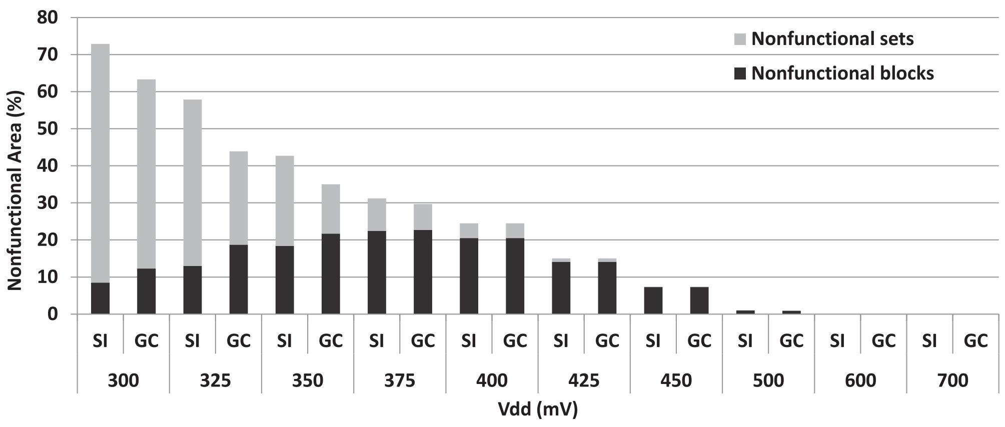


Fig. 12. L1 nonfunctional area for SI and GC remapping algorithms.

MGB = 2 for associativity = 8. By considering the same process for L2, as is obvious in Figure 12, we select the design point with either 8b or 16b subblock size and MGB = 1 for associativity = 4. We select the design point with 16b subblock size and MGB = 2 associativity = 8.

By selecting the parameter values mentioned previously at Vdd = 375 mV, the FDM overhead is 7% and 6.6% for L1 and L2, respectively. To reach such a low voltage level, FFT-Cache has sacrificed less than 25% of L1 cache blocks and almost 0% of L1 cache lines. For L2, FFT-Cache has sacrificed less than 20% of block and less than 4% of all L2 cache lines. In the next section, we will study the impact of FFT-Cache on power and performance.

5.4. Remapping Algorithm Analysis

In this section, we study the effect of FFT-Cache remapping algorithm on effective cache capacity. Figure 12 and 13 show the percentage of nonfunctional area of the cache, which is the sum of nonfunctional sets and blocks across different voltages for both the simple iterative (SI) and graph coloring (GC) algorithm of the remapping process. Figure 12 shows that for the voltages below 400 mV in L1, remapping with the graph coloring algorithm gains lower nonfunctional area than the one with simple remapping. For higher voltages, both schemes are almost the same (e.g., at 450 mV for L2 (Figure 13)). Another observation is that although GC slightly increases the amount of nonfunctional blocks compared to SI, it saves much more in nonfunctional

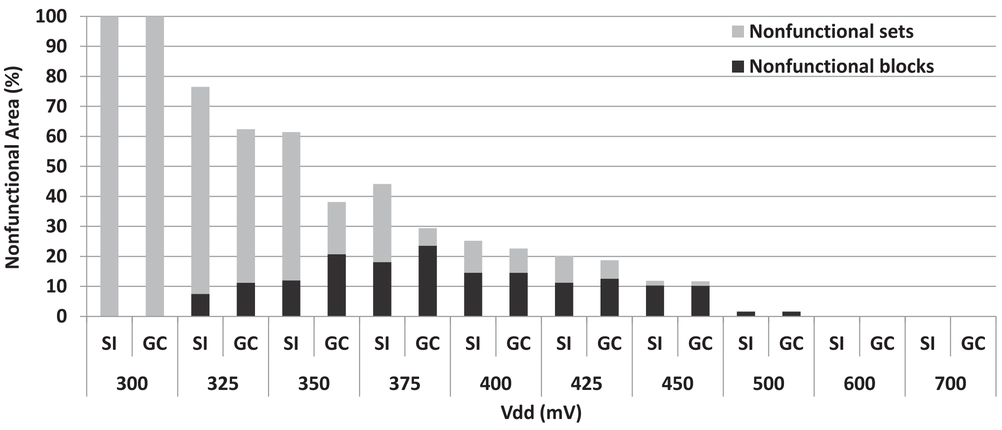


Fig. 13. L2 nonfunctional area for SI and GC remapping algorithms.

sets—that is, GC finds a better match for target blocks (either local or global) and uses them more efficiently than SI for fault replication.

When comparing SI and GC across both L1 (Figure 12) and L2 (Figure 13), we see that our graph coloring approach GC is more effective for L2, as the difference between nonfunctional areas between the two algorithms is generally higher for L2, because in L2 the blocks are larger (128B) and associativity is higher than L1. Consequently, this causes a higher degree of conflict among different blocks of each set (as seen in Figure 2), providing a better opportunity for GC compared to SI.

5.5. Performance Overhead

In Figure 14, we report the impact of FFT-Cache on processor performance when running SPEC2K benchmarks. The relative performance degradation in terms of instructions per cycle (IPC) is reported for a 64 KB four-way set associative L1 FFT-Cache (Figure 14(a)) and 2 MB eight-way set associative L2 FFT-Cache (Figure 14(b)). We also report the performance degradation when FFT-Cache scheme is deployed simultaneously in L1 and L2 (Figure 14(c)). Furthermore, we report the breakdown of performance drop, either due to increasing in cache access delay (from 2 to 3 cycles for L1 and 20 to 22 cycles for L2) or reduction in cache effective size.

The results are acquired for the minimum voltage configuration (MGB = 1, subblock size = 16 for L1 and MGB = 4, subblock size = 16 for L2). On average, performance drops by 2.2% for L1 cache and 1% for L2 cache. For L1 cache, the additional delay of accessing the cache is responsible for most performance loss. The impact of additional delay on performance is lower for L2 cache mainly due to the large L2 cache access delay (2 cycles delay overhead compared to 20 cycles L2 cache access delay). The results also indicate that the performance degradation for both L1 and L2 varies significantly across different benchmarks. The highest performance loss is observed in bzip2 and gzip benchmarks (more than 5% IPC loss). In fact, these are high IPC benchmarks. In these benchmarks, 1 cycle additional delay of L1 cache access in addition to reduction of L1 cache effective size by 20% is not tolerated. In many benchmarks, almost no performance loss is reported. These benchmarks include facerec, galgel, and lucas. Our investigation indicated that although in these benchmarks the cache miss rate increased slightly due to cache effective size reduction, the nature of out-of-order execution helped the benchmark to maintain the performance.

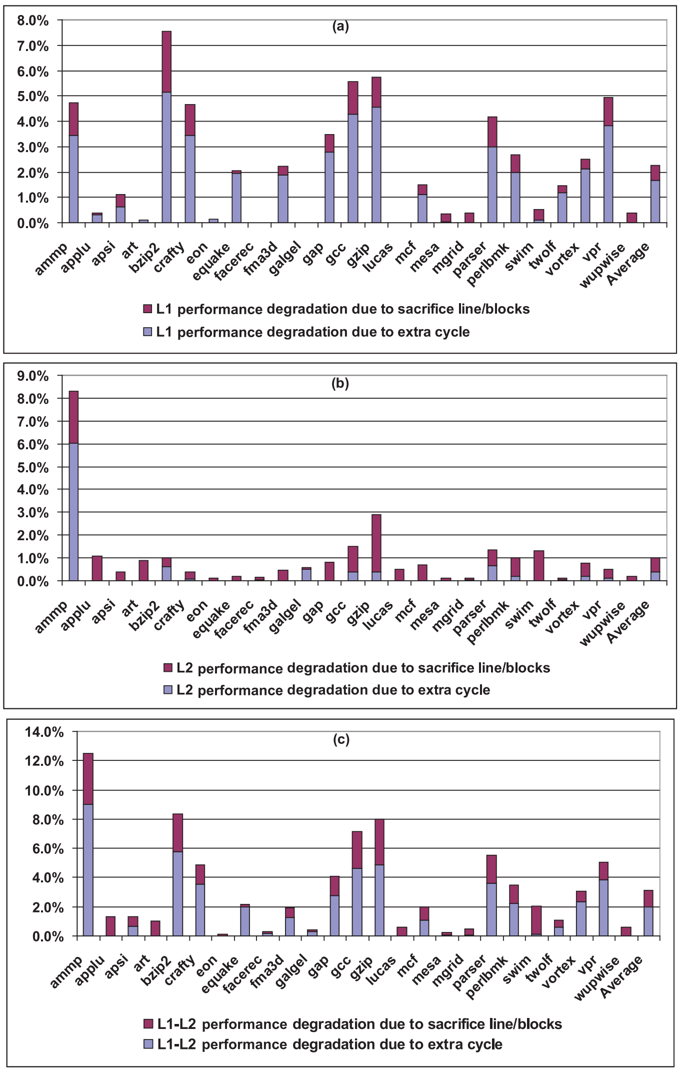


Fig. 14. Processor performance degradation when applying FFT-Cache for L1 (a), L2 (b), and L1 and L2 at the same time (c).

For L2 cache, the performance degradation is lower. The largest performance drop is 8% for the ammp benchmark. Finally, a 3% performance loss is observed when FFT-Cache scheme is deployed in both L1 and L2 caches.

5.6. Power and Area Overhead

Figure 15 summarizes the overhead of our scheme for both L1 and L2 caches. The power overhead in this figure is for high power mode (nominal Vdd). We account for the overheads of using 8T SRAM cells [Verma and Chandrakasan 2008] for protecting the tag and defect map arrays in low power mode. To reduce the effect of leakage and dynamic power consumption of FDM in high power mode, we assume that clock gating and power gating is applied in the FDM array. Therefore, the main source of dynamic

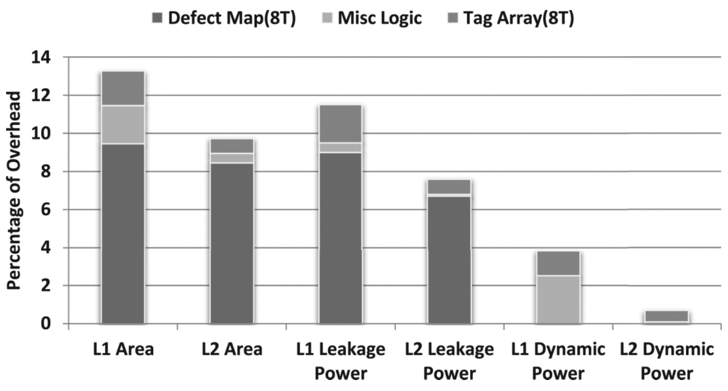


Fig. 15. Power and area overheads of FFT-Cache.

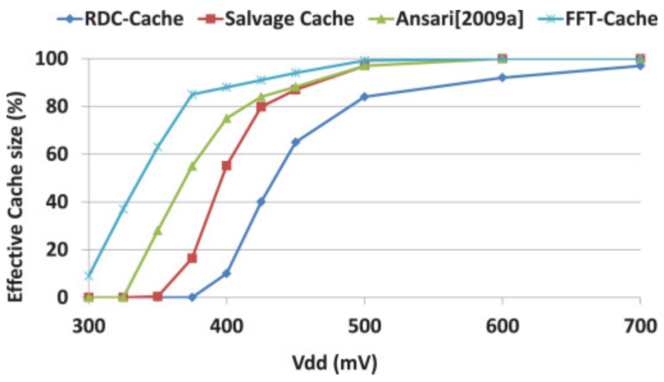


Fig. 16. Effective cache size for different fault-tolerant techniques.

power in nominal Vdd relates to bypass MUXs. As is showed in this figure, it is less than 3% for L1 but is trivial for L2.

As is evident in Figure 15, the defect map area is a major component of area overhead for both L1 and L2. The total area overhead for L1 is 13% and for L2 is less than 10%. In addition, the defect map is the major component of leakage power in both L1 and L2.

Based on our CACTI results in both nominal Vdd (660 mV) and Vdd-min (375 mV), we achieve 66% dynamic and 48% leakage power reduction in L1 cache and 80% dynamic and 42% leakage power reduction in L2 cache.

5.7. Comparison to State-of-the-Art Methods

To illustrate the effectiveness of FFT-Cache, first we compare the efficacy of our methodology in fault remapping against multiple state-of-the-art remapping-based fault-tolerant techniques proposed in recent years. Then we perform a quantitative comparison of FFT-Cache to four well-known alternative cache protection schemes.

5.7.1. Comparison to Recent Remapping Techniques. Inthis section, we compare FFT-Cache to three state-of-the-art remapping techniques: RDC-Cache [Sasan et al. 2009], Salvage Cache [Koh et al. 2009a], and Ansari's work [Ansari et al. 2009a]. Figure 16 shows the effective cache size for each of these techniques across various voltage points. We use the failure rates reported in Sasan et al. [2010] for 65 nm technology node. The results reported in this figure are based on a 1,000-run Monte Carlo simulation for an 8MB eight-way set associative LLC cache with two banks and 64 bytes block size. To

Table II. Comparison of Different Cache Fault-Tolerant Schemes

Scheme	Vdd-Min (mV)	L1 Cache Area over. (%)	L1 Cache Power over. (%)	L2 Cache Area over. (%)	L2 Cache Power over. (%)	Norm. IPC
6T cell	660	0	0	0	0	1.0
ZerehCache	430	16	15	8	12	0.97
Wilkerson	420	15	60	8	20	0.89
Ansari	420	14	19	5	4	0.95
10T cell	380	66	24	66	24	1.0
FFT-Cache	375	13	16	10	8	0.95

have a fair comparison, for all of these techniques, including FFT-Cache, we assume that fault map area overhead is at most 9% of the cache size (the area overhead of FFT-cache with 16-bit subblocks). Based on this assumption and to meet the overhead limit, Ansari's method selects 16-bit subblock size. The subblock size for RDC-Cache and Salvage-Cache are 128-bit and 64-bit, respectively. The results show that in ultra-low voltage region (below 400 mV), as the failure rate increases, FFT-Cache results in much smaller cache capacity loss.

5.7.2. Quantitative Comparison to Alternative Methods. In this section, we compare our scheme to four state-of-the-art works: Wilkerson et al. [2008], ZerehCache [Ansari et al. 2009b], 10T SRAM cell [Calhoun and Chandrakasan 2006], and Ansari et al. [2009a]. This is just a quantitative comparison to compare the overheads of different schemes while providing a rough comparison of their min-Vdd. We obtained these overhead results either directly from the related papers (when a direct comparison was possible) or by trying to recalibrate the results to enable a fair comparison for the same technology node. We calculated the Vdd-min for ZerehCache and updated the Vdd-min of Wilkerson's scheme using our cache configuration and fault model in 90nm. All other overheads were obtained directly from the papers. Note that except for 10T cell and Wilkerson's work, which use 65nm technology, all other schemes, including FFT-Cache, use 90nm. A 380 mV Vdd-min for the 10T cell was derived in 65nm, and it is clear that in 90nm this value would be higher. Table II summarizes this comparison based on the minimum achievable Vdd, area, and power overheads for both L1 and L2 caches, and normalized IPC. In this table, different techniques are sorted based on the minimum achievable Vdd-min when targeting 99.9% yield.

Overall, our proposed FFT-Cache achieves the lowest operating voltage (375 mV) and the highest power reduction compared to all other techniques. The closest techniques to ours are 10T cell, Ansari, and Wilkerson. The 10T cell achieves almost similar power reduction to FFT-Cache but incurs a 66% area overhead. Wilkerson's work has a significant power overhead and also suffers an 11% performance degradation. The Ansari technique incurs slightly lower power and area overhead just for L1 cache compared to FFT-Cache, but it does not reduce operating voltage below 420 mV and thus achieves lower power reduction. Overall, the FDM of FFT-Cache along with high configurability and high flexibility allow it to tolerate higher failure rates compared to other similar techniques.

6. RELATED WORK

Several fault-tolerant techniques have been proposed at multiple levels of abstraction to improve the cache yield and/or lower the minimum achievable voltage scaling bound.

6.1. Circuit-Level Techniques

A number of research efforts use circuit-level techniques to improve the reliability of each SRAM cell. Besides the conventional 6T SRAM cell, several other designs, including 8T SRAM cell [Chang et al. 2005; Chen et al. 2007], 10T SRAM cell [Calhoun and Chandrakasan 2006], and 11T SRAM cell [Moradi et al. 2008], have been proposed. All of these SRAM cells improve read stability, although the stability of the inverter pair remains unchanged. Most of these cells incur a large area overhead that poses a significant limitation for performance and power consumption of caches. For instance, Kulkarni et al. [2007] proposed a Schmidt trigger-based 10T SRAM cell with inherent tolerance toward process variation using a feedback-based mechanism. However, this SRAM cell requires a 100% increase in the area and about a 42% increase in the access time for low voltage operation.

6.2. Error Coding Techniques

At the system level, a wide range of error detection codes (EDCs) and error correcting codes (ECCs) have been studied. ECC is proven as an effective mechanism for handling soft errors [Chen and Hsiao 1984]. However, in a high-failure rate situation, most coding schemes are not practical because of the strict bound on the number of tolerable faults in each protected data chunk. In addition, using ECCs incurs a high overhead in terms of storage for the correction code, large latency, and slow and complex decoding [Kim et al. 2007]. A recent effort uses a configurable part of the cache for storing multiple ECC check bits for different segments of cache line using an elaborate Orthogonal Latin Square Code ECC [Chishti et al. 2009] to enable dynamic error correction. This requires up to eight levels of XOR gates for decoding, resulting in a significant increase in cache critical path delay. Yoon et al. [2009a, 2009b] leverage a two-tiered ECC scheme by decoupling error correction from error detection and using offline memory to store error correction data to reduce area and power overheads of last-level cache (LLC) protection.

6.3. Architecture-Level Techniques

Several architectural techniques have also been proposed to improve reliability of on-chip cache by using either redundancy or cache resizing. Earlier efforts on fault-tolerant cache design use various cache down-sizing techniques by disabling a faulty line or block of cache. Ozdemir et al. [2006] proposed Yield-Aware cache, in which they developed multiple techniques that turn off either cache ways or horizontal regions of the cache that cause delay violation and/or have excessive leakage. Agarwal et al. [2005] proposed a fault-tolerant cache architecture in which the column multiplexers are programmed to select a nonfaulty block in the same row if the accessed block is faulty. Similarly, PADed cache [Shirvani and McCluskey 1999] uses programmable address decoders that are programmed to select nonfaulty blocks as replacements of faulty blocks. Sasan et al. [2009, 2010] proposed a number of cache architectures in which the error-prone part of the cache is fixed using either a separate redundancy cache or parts of the same cache. RDC-Cache [Sasan et al. 2009] replicates a faulty word by another clean word in the last way of the next cache bank.

Wilkerson et al. [2008] propose two schemes: Word-disable (WDIS) and Bit-fix (BFIX). The WDIS scheme combines two consecutive cache blocks into a single cache block, thereby reducing the capacity by 50%, whereas the BFIX scheme sacrifices a (functional) cache block to repair defects in three other cache blocks, thereby reducing the capacity by 25%.

The buddy cache [Koh et al. 2009b] pairs up two nonfunctional blocks in a cache line to yield one functional block. A similar idea was proposed independently in [Roberts et al. 2007]. The salvage cache [Koh et al. 2009a] improves on this technique by using a single nonfunctional block to repair several others in the same line. However, all of these methods are not efficient in the near-threshold region with high fault probabilities. Indeed at such a low voltage, the cache would be effectively downgraded to just a fraction (e.g., 30%) of its original size, which results in a large performance degradation in terms of IPC across standard benchmarks.

ZerehCache [Ansari et al. 2009b] introduces an interconnection network between the row decoder and data array, which requires significant layout modifications. In this scheme, an external spare cache is used to provide redundancy; thus, applying the interconnection network allows limited redundancy borrowing across the statically specified, fixed-size groups. However, its interconnection network has a noticeable area overhead and power consumption cost.

MC^2 [Chakraborty et al. 2010, 2013] maintains multiple copies of each data item, exploiting the fact that many embedded applications have unused cache space resulting from small working set sizes. On every cache access, MC^2 detects and corrects errors using these multiple copies. Thus, MC^2, although particularly useful for embedded applications with small working sets, may result in high area and performance overhead for other applications, particularly in the presence of high fault rates.

Ansari et al. [2009a] propose a fault-tolerant cache that intertwines a set of $n + 1$ partially functional cache lines together to give the overall appearance of n functional lines. They partition the set of all cache word lines into large groups, where one word line (the sacrificial line) from each group is set aside to serve as the redundant word line for the other word lines in the same group.

Our proposal differs from previous approaches in several aspects. First, FFT-Cache is more efficient in remapping of faulty blocks compared to other methods: we categorize the faulty cache lines based on the degree of conflicts among their blocks and use this conflict information for efficient remapping. Second, FFT-Cache uses an FDM that simplifies the mapping and remapping of faulty areas; in addition, the FDM overhead is independent of the fault rate. Furthermore, by configuring the MGB parameter, we can trade the associativity to do more remapping and save more cache capacity when the amount of conflicting faulty blocks is high. Finally, FFT-Cache leverages a graph coloring algorithm to configure its FDM to minimize the amount of nonfunctional parts of the cache.

7. CONCLUSIONS

In this work, we proposed a fault-tolerant cache architecture—FFT-Cache—which has an FDM to configure its architecture to achieve significant reduction in power consumption through aggressive voltage scaling while maintaining high error reliability. FFT-Cache uses a portion of faulty cache blocks as redundancy to tolerate other faulty cache lines and blocks. This is accomplished by using either block- or line-level replication in the same set or between two or more sets. FFT-Cache has an efficient configuration algorithm that categorizes the cache lines based on degree of conflict between their blocks to reduce the granularity of redundancy replacement. Using our approach, the operational voltage of the cache is reduced down to 375mV in 90nm technology. This achieves 66% and 80% dynamic power reduction for L1 and L2 caches, respectively. FFT-Cache reduces the leakage power of L1 and L2 caches by 48% and 42%, respectively. This significant power saving comes with a small 5% performance loss and 13% area overhead.

REFERENCES

A. Agarwal, B. C. Paul, H. Mahmoodi, A. Datta, and K. Roy. 2005. A process-tolerant cache architecture for improved yield in nanoscale technologies. *IEEE Transaction on VLSI Systems* 13, 1, 27–38.

H. Al-Omari and K. Sabri. 2006. New graph coloring algorithms. *American Journal of Math and Statistics* 2, 4, 739–741.

A. Ansari, S. Feng, S. Gupta, and S. Mahlke. 2009a. Enabling ultra low voltage system operation by tolerating on-chip cache failures. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. 307–310.

A. Ansari, S. Gupta, S. Feng, and S. Mahlke. 2009b. ZerehCache: Armoring cache architectures in high defect density technologies. In *Proceedings of the International Symposium on Microarchitecture (Micro)*. 100–110.

T. Austin, E. Larson, and D. Ernst. 2002. SimpleScalar: An infrastructure for computer system modeling. *IEEE Transactions on Computers* 35, 2, 59–67.

B. Calhoun and A. Chandrakasan. 2006. A 256kb sub-threshold SRAM in 65nm CMOS. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)*. 2592–2601.

A. Chakraborty, H. Homayoun, A. Khajeh, N. Dutt, A. Eltawil, and F. Kurdahi. 2010. E < MC2: Less energy through multi-copy cache. In *Proceedings of the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES)*. 237–246.

A. Chakraborty, H. Homayoun, A. Khejah, N. Dutt, A. Eltawil, and F. Kurdahi. 2013. Multicopy cache: A highly energy-efficient cache architecture. *ACM Transactions on Embedded Computing Systems* 13, 5, Article No. 150.

L. Chang, D. M. Fried, J. Hergenrother, J. W. Sleight, R. H. Dennard, R. K. Montoye, L. Sekaric, S. J. McNab, A. W. Topol, C. D. Adams, K. W. Guarini, and W. Haensch. 2005. Stable SRAM cell design for the 32 nm node and beyond. In *Proceedings of the Symposium on VLSI Technology*. 128–129.

C. Chen and M. Hsiao. 1984. Error-correcting codes for semiconductor memory applications: A state of the art review. *IBM Journal of Research and Development* 28, 2, 124–134.

G. Chen, D. Blaauw, T. Mudge, D. Sylvester, and N. Kim. 2007. Yield-driven near-threshold SRAM design. In *Proceedings of the International Conference on Computer Aided Design*. 660–666.

Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu. 2009. Improving cache lifetime reliability at ultra-low voltages. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42)*. 88–99.

J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. C. Hoe. 2007. Multi-bit error tolerant caches using two-dimensional error coding. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40)*. 197–209.

W. Klotz. 2002. *Graph Coloring Algorithms*. Mathematik-Bericht 5, Clausthal University of Technology, Clausthal, Germany.

C. K. Koh, W. F. Wong, Y. Chen, and H. Li. 2009a. The salvage cache: A fault-tolerant cache architecture for next-generation memory technologies. In *Proceedings of the International Conference on Computer Design (ICCD)*. 268–274.

C. K. Koh, W. F. Wong, Y. Chen, and H. Li. 2009b. Tolerating process variations in large, set associative caches: The buddy cache. *ACM Transactions on Architecture and Code Optimization* 6, 2, Article No. 8.

J. P. Kulkarni, K. Kim, and K. Roy. 2007. A 160 mv, fully differential, robust Schmitt trigger based sub-threshold SRAM. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 171–176.

F. Moradi, D. Wisland, S. Aunet, H. Mahmoodi, and T. Cao. 2008. 65 nm sub-threshold 11t-SRAM for ultra low voltage applications. In *Proceedings of the International Symposium on System-on-a-Chip*. 113–118.

Y. Morita, H. Fujiwara, H. Noguchi, Y. Iguchi, K. Nii, H. Kawaguchi, and M. Yoshimoto. 2007. An area-conscious low-voltage-oriented 8t-SRAM design under DVS environment. In *Proceedings of the IEEE Symposium on VLSI Circuits*. 256–257.

S. Mukhopadhyay, H. Mahmoodi, and K. Roy. 2005. Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24, 12, 1859–1880.

N. Muralimanohar, R. Balasubramonian, and N. Jouppi. 2009. Cacti 6.5. Technical Report. HP Laboratories.

S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou. 2006. Yield-aware cache architectures. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. 15–25.

D. Roberts, N. S. Kim, and T. Mudge. 2007. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology. In *Proceedings of the 10th Euromicro Conference on Digital System Design (DSD)*. 570–578.

A. Sasan, H. Homayoun, A. Eltawil, and F. Kurdahi. 2009. A fault tolerant cache architecture for sub 500mV operation: Resizable data composer cache (RDC-cache). In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. 251–260.

A. Sasan, H. Homayoun, A. M. Eltawil, and F. Kurdahi. 2010. Inquisitive defect cache: A means of combating manufacturing induced process variation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 18, 12, 1–13.

P. P. Shirvani and E. J. Mccluskey. 1999. PADded cache: A new fault-tolerance technique for cache memories. In *Proceedings of the 17th IEEE VLSI Test Symposium (VTS'99)*. 440–445.

N. Verma and A. Chandrakasan. 2008. A 256 kb 65 nm 8t subthreshold SRAM employing sense-amplifier redundancy. *IEEE Journal of Solid-State Circuits* 43, 1, 141–149.

C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. 2008. Trading off cache capacity for reliability to enable low voltage operation. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA)*. 203–214.

W.-F. Wong, C.-K. Koh, Y. Chen, and H. Li. 2007. VOSCH: Voltage scaled cache hierarchies. In *Proceedings of the 25th International Conference on Computer Design*. 496–503.

D. H. Yoon and M. Erez. 2009a. Memory mapped ECC: Low-cost error protection for last level caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*. 116–127.

D. H. Yoon and M. Erez. 2009b. Flexible cache error protection using an ECC FIFO. In *Proceedings of the Conference on High Performance Computing, Networking, Storage, and Analysis (SC'09)*. Article No. 49.

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

Received June 2013; revised January 2014; accepted May 2014.