FFT-Cache: A Flexible Fault-Tolerant Cache Architecture for Ultra Low Voltage Operation

Abbas BanaiyanMofrad[1], Houman Homayoun[2], Nikil Dutt[1]
[1]Center for Embedded Computer Systems
University of California, Irvine
[2]Department of Computer Science and Engineering
University of California, San Diego
{abanaiya, dutt}@uci.edu, hhomayou@eng.ucsd.edu

ABSTRACT
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 paper, we propose Flexible Fault-Tolerant Cache (FFT-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 of 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 SPEC2K benchmarks demonstrate that the operational voltage can be reduced down to 375mV, which achieves up to 80% reduction in dynamic power and up to 48% reduction in leakage power with small performance impact and 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
Algorithms, Design, Reliability

Keywords
Low power cache, Fault-tolerant cache, Flexible fault remapping

1. INTRODUCTION
Caches are already known to consume a large portion (about 30-70%) of total processor power [3][4] and on-chip cache size will continue to grow due to device scaling coupled with performance requirements. Therefore, it becomes critical to manage the power and reliability of the caches in order 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 [1][2]. Figure 1 represents the failure rate of an SRAM cell based on the operational voltage in a 90nm technology [14][30]. To save power while achieving an acceptable manufacturing yield of 99.9% for 64KB L1 and 2MB L2 caches, a minimal operational voltage must be selected. From Figure 1 we can see that the probability of failure for each memory array must be kept at less than 1 out of 1000 to get this yield. Based on this assumption, we estimate the minimum operational voltage for L1 as 620 mV and for L2 660 mV, and we are not able to further reduce the operational voltage without incurring cell failures.

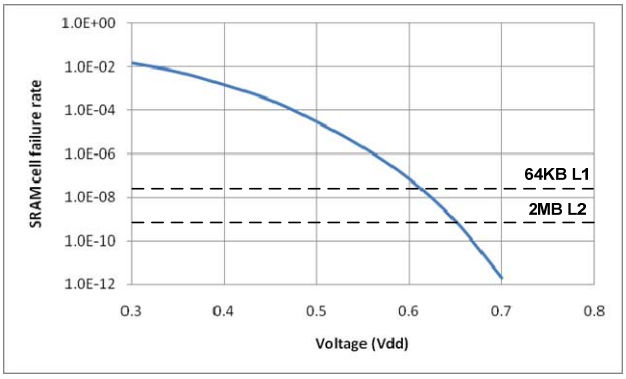
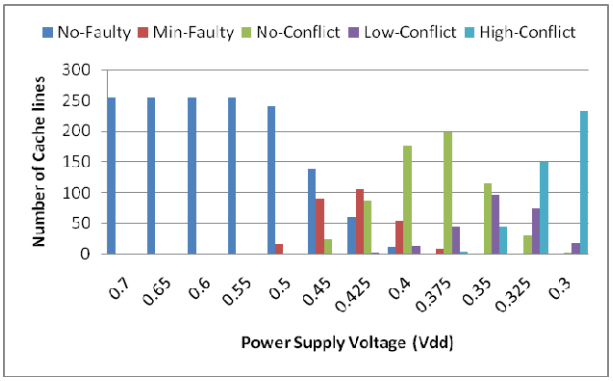


Figure 1.Probability of SRAM cell failure vs.V_{dd}

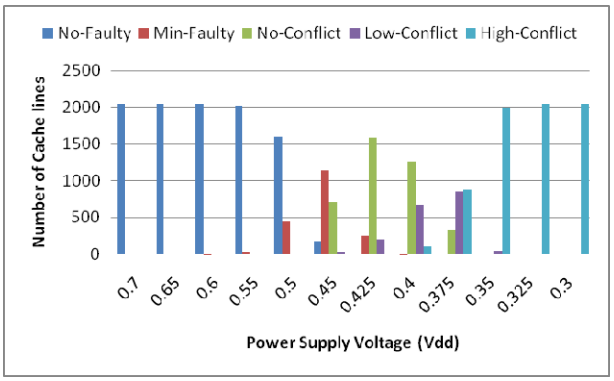
Since applications may not be tolerant to even a single bit error, typically caches must be operated at a high Vdd to ensure a very low probability of failure, leading to high energy consumption. However, by exploiting mechanisms that allow a cache to become inherently resilient to a large number of cell failures, we can operate the cache at a lower Vdd and thus gain significant energy savings. This paper presents FFT-Cache, an approach that aims to: 1) design a very low power, fault-tolerant cache architecture that can detect and replicate memory faults arising from operation in the near-threshold region; 2) minimize non-functional or redundant memory area to lessen impact on cache capacity; and 3) tolerate cache faults as much as possible.

CASES'11, October 9–14, 2011, Taipei, Taiwan.

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/L2 caches and the results are represented in Figure 2. Before description of this figure, let's define some parameters. For the rest of this paper, 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. Also, 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 blocks (lines) have a conflict if they have at least one faulty subblock (block) in the same position. Also, 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.



(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

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

Figure 2 represents the number of caches lines in different categories for a 64KB 4-way set associative L1 and a 2MB 8-way set associative L2 cache for different Vdd values. In this figure we categorize the cache lines to five groups:

No-Faulty -- include cache lines with no faulty block.

Min-Faulty -- include cache lines with the number of faulty blocks below a threshold, 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 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 will be increased. 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. To the best of our knowledge most of the previous cache protection techniques only consider conflicts between two or more cache lines and none of them deals with conflicts inside of the cache lines [18][19][28][30].

In this work, we propose Flexible Fault-Tolerant Cache (FFT-Cache), a cache architecture that uses a flexible defect map to efficiently tolerate the large number of faults when operating in the near threshold region. FFT-Cache uses a novel flexible defect map to replicate faulty data blocks in both the same set and different cache sets. It divides each cache block into multiple sub-blocks. FFT-Cache uses a portion of faulty cache blocks as redundancy to tolerate other faulty caches lines and blocks. This can be accomplished by using either block-level or line-level replication in the same set or between two or more sets. Our configuration algorithm categorizes the cache lines based on degree of conflict between their blocks to reduce the granularity of redundancy replacement. Using this approach, FFT-Cache first attempts to replicate faulty blocks in the same set; otherwise it attempts to use faulty blocks to replicate other faulty lines. If not, it attempts to disable the fewest number of cache lines for tolerating faulty blocks/lines. While significantly increasing fault-tolerance, our FFT-Cache approach when operated in low-power mode incurs a small (5%) performance degradation and a small (13%) area overhead.

The main contributions of our FFT-Cache approach are that we: 1) deploy a new flexible defect map to replicate faulty data blocks in both the same set and different cache sets; 2) use a portion of faulty cache lines (global blocks) as redundancy to tolerate other faulty blocks or lines; 3) categorize the cache lines based on the degree of conflict of their blocks to reduce the granularity of redundancy replacement; and 4) use a simple and efficient algorithm to initiate and update the flexible defect map to optimize the proposed fault-tolerant architecture with minimum non-functional cache area.

The rest of this paper is organized as follows: Section 2 reviews related work and distinguishes our proposed approach. Section 3 introduces the proposed FFT-Cache architecture. Section 4 evaluates the architecture and presents experimental results. Section 5 concludes the paper.

2. RELATED WORK

In the literature, several fault-tolerant techniques have been proposed to improve the cache yield and/or lower the minimum achievable voltage scaling bound. A number of these works use circuit-level techniques to improve the reliability of each SRAM cell. Besides the familiar 6T SRAM cell, several other designs, including 8T SRAM cell [8][9], 10T SRAM cell [13], and 11T SRAM cell [11] have been proposed. All of these SRAM cells

improve read stability, though the stability of the inverter pair remains unchanged. Most of these cells have a large area overhead which poses a significant limitation for performance and power consumption of caches. Kulkarni et al. [12] proposed a Schmidt trigger based 10T SRAM cell with inherent tolerance towards process variation using a feedback-based mechanism. However, this SRAM cell requires a 100% increase in area and about 42% increase in access time for low voltage operation.

At the system level, a wide range of Error Detection Code (EDC) and Error Correcting codes (ECC) have been used. ECC is proven as an effective mechanism for handling soft errors [23]. 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 ECC codes incurs a high overhead in terms of storage for the correction code, large latency, slow and complex decoding [16]. A recent work 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 [21] to enable dynamic error correction. This requires up to 8 levels of XOR gates for decoding, resulting in significant increase in cache critical path delay.

Several architectural techniques have also been proposed to improve reliability of on-chip cache by using either redundancy or cache resizing. Earlier works on fault-tolerant cache design use various cache down-sizing techniques by disabling a faulty line or block of cache. Ozdemiret et al. proposed Yield-Aware cache [15] 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. [1] proposed a fault tolerant cache architecture in which the column multiplexers are programmed to select a non-faulty block in the same row, if the accessed block is faulty. Similarly, PADed cache [17] uses programmable address decoders that are programmed to select non-faulty blocks as replacements of faulty blocks. Sasan et al. [19][20] 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 [19] replicates a faulty word by another clean word in the last way of next cache bank.

In [18] two schemes called Word-disable (WDIS) and Bit-fix (BFIX) have been proposed. 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 [27] pairs up two non-functional blocks in a cache line to yield one functional block. A similar idea was proposed independently in [26]. The salvage cache [28] improves on this technique by using a single non-functional block to repair several others in the same line [28]. 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 [29] 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 a 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 [22] 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 – while 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. [30] 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.

FFT-Cache differs from previous approaches in that it minimizes the amount of non-functional area by using global blocks inside of functional lines to keep the replication data, resulting in lower area and power overhead.

3. PROPOSED ARCHITECTURE

In this section, we first describe the proposed FFT-Cache architecture that uses a Flexible Defect Map (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

The FFT-Cache architecture has two banks of data that can be accessed concurrently, and 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 the minimal number of cache lines to minimize performance degradation and tolerate the maximum amount of defects. 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 a Flexible Defect Map (FDM) which is then used to configure the address mapping. 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*. If FFT-Cache cannot find a target block for the host line, 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: 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. In order 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. Note that since target blocks/lines do not store any independent data, they are considered non-functional. 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 we minimize the number of non-functional and target lines by initially attempting

to find a local target block to sacrifice. If a local target block was not found, it 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 other fault-tolerant methods do.

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 is 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 the FFT-Cache.

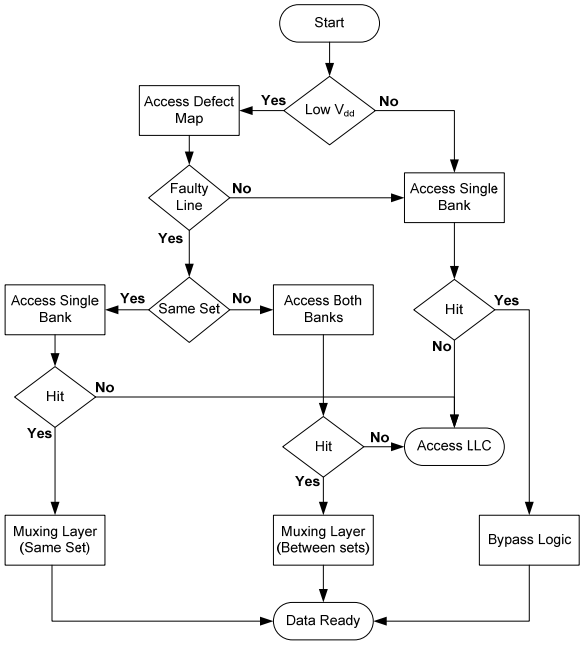


Figure 3. FFT-Cache Access Flowchart

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 memory Built-In Self Test (BIST) unit, the L1 and L2cache(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, in order to protect the defect map and the tag arrays, we use the well studied 8T SRAM cell [10] 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 memory structures without failing. An example of an FDM entry for a cache with associativity of 4 is represented in Figure 4.

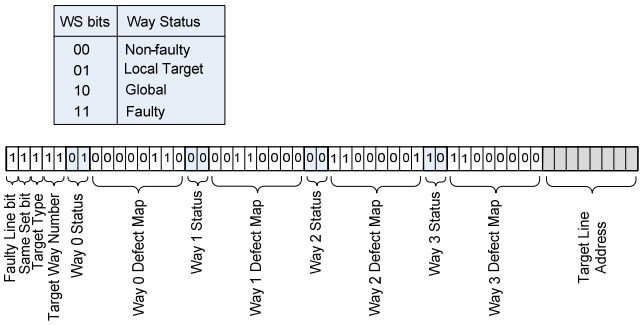


Figure 4. Details of a row of FDM

Each FDM entry includes multiple configuration bits, the defect map of each way (block), 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) *Non-Faulty*, 2) *Faulty*, 3) *Local Target*, and 4) *Global*. Initially the status of all blocks in the cache is Non-faulty, representing the absence of any faulty subblocks. If a block contains at least one faulty subblock, its status will be 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 a set and cannot be used as a local target block, gets the status of Global and may be used as a Global target block. As mentioned before, MGB represents the maximum number of blocks in a line that can be set as Global block; the remaining blocks can then be composed as a group of blocks without conflict, which allows them to find a Global Target block. This would guarantee that at most half of the cache needs to be sacrificed for tolerating faults.

We now present the algorithms for FDM initialization and configuration.

Begin FDM initialization algorithm

1. *Run BIST and find faulty cache lines at a subblock level and fill defect map sections of each entry*
2. *Set the "Way Status" field of faulty blocks to 11(Faulty) and other non-faulty blocks to 00(Non-faulty)*
3. *For lines with at least one faulty subblock, set the "Faulty Line" bit to 1 (Faulty) and for other lines with no faulty subblock, set it to 0 (Non-faulty)*
4. *Set "Target Line" address field of all rows same as their address*
5. *If the number of faulty blocks in a set is below MGB then set "Same Set" bit to 0*
6. *If there is no conflict between all except MGB blocks in a line then:*
   - *Set "Same Set" bit to 1 and make them a group and select one of them as Target block*
   - *Set "Way Status" bits of the Target block to 01(Target) and set its number in the "Target Way" number*
   - *For other out of group blocks (if any), set their status bits to 10(Global)*
7. *If there are more than one conflict between blocks then set Same Set bit to 0*
8. *Repeat Steps 5-8 for all entries of FDM*

End FDM initialization algorithm

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

Begin FDM configuration algorithm

1. *Traverse the faulty rows of FDM and based on the conflicts between faulty blocks in each row, categorize the FDM entries into four groups:*
   - *If the number of faulty blocks in a row is below MGB set it in the min_faulty block group.*
   - *If there is no conflict between faulty blocks in a row, set the row in the no_conflict group.*
   - *If there is only one conflict between one of the blocks with other blocks in the row, set the row in the low_conflict group.*
   - *If there is more than one conflict between the blocks within a group, set the row in the high_conflict group.*
2. *For lines in the min_faulty block group set the status of faulty blocks to Global block.*
3. *For lines in the no_conflict group make one of its faulty blocks as Local Target block.*
4. *For the lines in low_conflict group, make the status of the block that has conflict with other blocks as Global and then attempt to find a Global Target block for each line.*
5. *For the lines in high_conflict group, try to find a similar line from other bank to make it as the Global Target line.*

End FDM configuration algorithm

Figure 5 shows an example of the FDM configuration for a given distribution of faults in 5 lines of a 4-way set associative cache with 4 subblocks in each block (way). The first line is a clean line without any faulty blocks. The second line is a member of min_faulty group and sets its faulty block as Global block. The third line is an example of a no_conflict group in which the first block (in Way0) is set as Local Target block to be sacrificed for fault-tolerance of other faulty blocks. The fourth line is a member of the low_conflict group with the first block as a Global block which has conflict with other blocks. The fifth line is a member of high_conflict group with two conflicts between its blocks (block0 has conflict with block2 and block1 with block3).

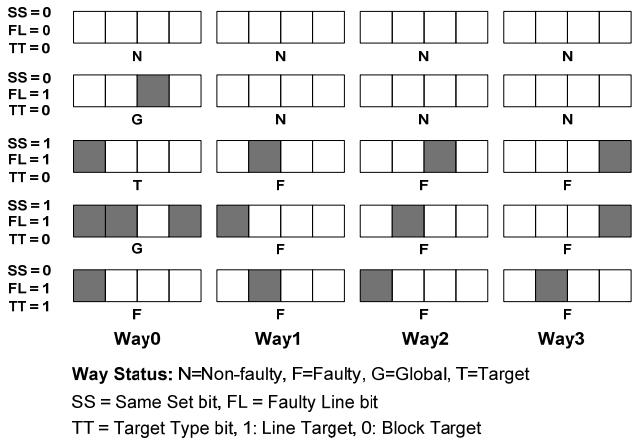


Figure 5. An example of FDM configuration for a given distribution of faults in a 4-way set associative cache.

3.3 Architecture Details

We now present the architecture of the FFT-Cache. We begin with the architecture of a conventional 2-way set associative cache (labeled CC) in Figure 6. Based on the tag match results, either Way0 or Way1 is being selected. The data is transferred to/from memory cells using multiplexers as indicated in the figure.

Figure 7 shows the architecture of the proposed FFT-Cache. Let's assume the cache is divided into two banks, with each bank containing two ways (blocks) that are further divided into 2 subblocks. The new modules (MUXs and FDM) added to the conventional cache are highlighted in the figure. Note that two levels of MUXing are added to compose the final fault-free block, based on either multiple blocks within a set or between two or more sets in different banks of data. The additional multiplexer network would allow us to compose the final fault-free block from any of the subblocks in any of the cache way in either Bank0 or Bank1, based on the FDM data. Next, we present the hardware implementation of the FFT-Cache and analyze its overhead.

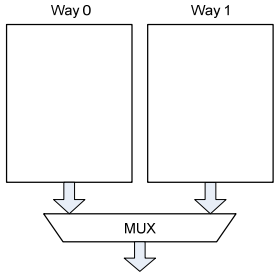


Figure 6. A conventional 2-way set associative cache (CC).

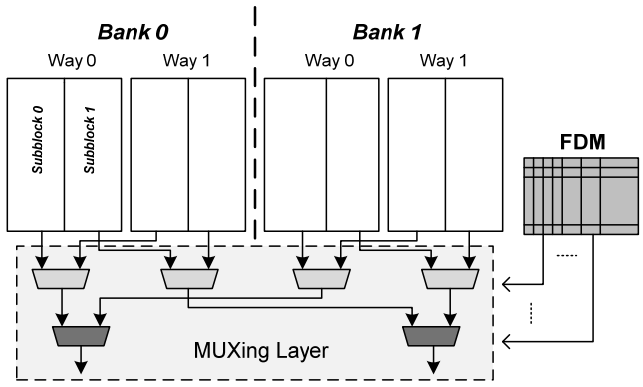


Figure 7. Architecture details of the proposed FFT-Cache with FDM and 2 subblocks per block.

The total number of n-to-1 multiplexers required to compose the final fault-free block is: $k \times (n \times b - 1)$

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 2 bank, 64KB, 4-way set associative cache, with each way with 16 subblocks, a total of 112 4-to-1 multiplexers are required. The size of FDM equals to the multiplication of number of cache lines by FDM entry size. For such cache, the FDM size will be $256 \times [5 + 4 \times (2 + 32) + 8]$ bits.

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 Synopsys Design Compiler for TSMC 90nm 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 while area and power consumption are increased by only 2% compared to a CC MUX network.

Recall that the 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 overheads as low as possible. Before presenting the experimental results, we describe our methodology/experimental set-up, develop an analytical failure model, and outline the exploration space for our experiments.

4.1 Methodology

Table 1 outlines our experimental setup for the baseline processor configuration. The processor is configured with a 64K 4-way set associative L1 cache. The architecture was simulated using an extensively modified version of Simplescalar 4.0 [5] 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 the reference data sets. We used CACTI6.5 [7] to evaluate area, power and delay of both L1 and L2 caches and their related FDMs. The Synopsys standard industrial tool-chain (with TSMC 90nm 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 2 cycles is assumed to be broken into actual cache access taking place in cycle 1, while 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%) in 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.

Table 1.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

For a given set of cache parameters (e.g., associativity, subblock size, MGB, etc.), a Monte Carlo simulation with 1000 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 Cache Failure

To estimate the probability of failure for the cache, we developed an analytical model of the FFT-Cache. 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. We also define c as the maximum acceptable disabled blocks in a set (MGB). We derive the following equations:

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

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

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

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

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:

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

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:

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

The probability of two cache sets being paired with no conflict such that at least one of β possible groups of **γ** blocks in them is paired:

$$P_{\text{paired-set}} = P_{ps}= \beta\,(P_{pb})^{\gamma}, \gamma = n-c,\ \beta = \binom{n}{n-c}$$

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

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

Let's consider R as the number of cache lines that can be disabled, so at most 2R sets can be faulty but paired for the cache to be operational. The probability that the FFT-Cache is operational is:

$$P_{Cac\square e} = \sum_{i=0}^{R} \binom{m}{2i} P_{set}^{m-2i}(1-P_{set})^{2i}\,P_{ps}^{i}$$

We used our analytical models of failure to determine the failure probability of a 64KB L1 FFT-Cache and compared it to SECDED and DECTED methods with equal area overhead. The results, as shown in Figure 8, demonstrate that, at a given voltage, FFT-Cache is the most reliable cache, while SECDED is the least reliable cache. If we adopt the definition for Vdd-min as the voltage at which 1 out of every 1000 cache instances is defective [18], based on this figure the FFT-Cache can reduce the Vdd below 375mv in comparison with 465mv and 520mv for DECTED and SECDED methods, respectively.

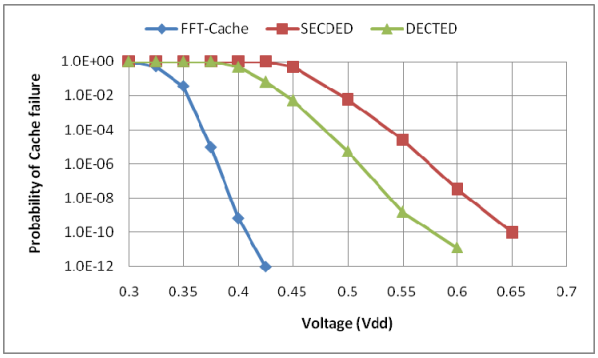
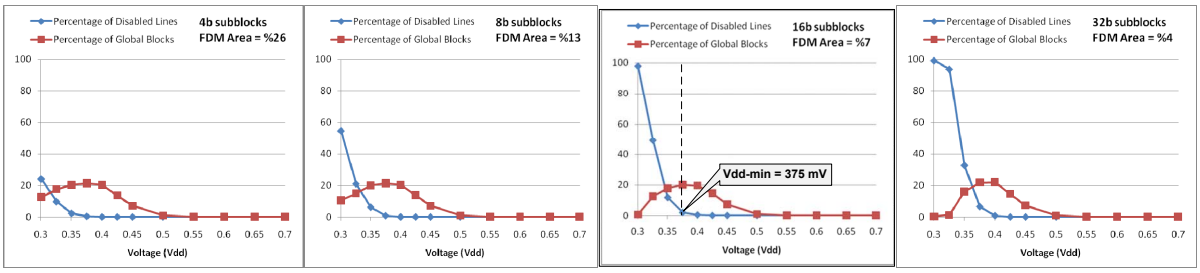
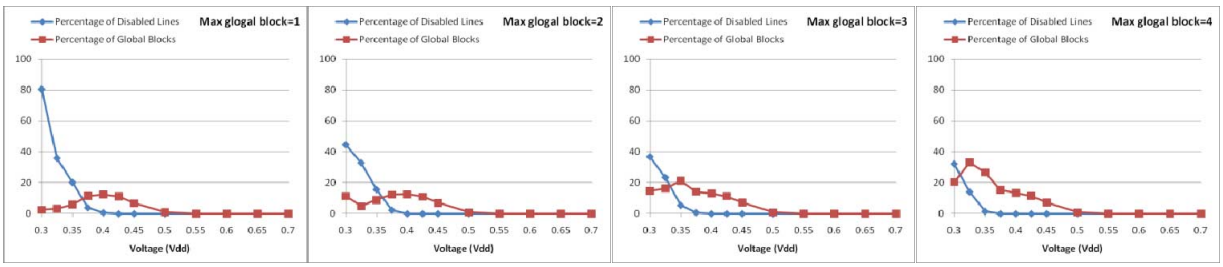


Figure 8. Probability of cache failure vs Vdd for SECDED, DECTED, and FFT-Cache.

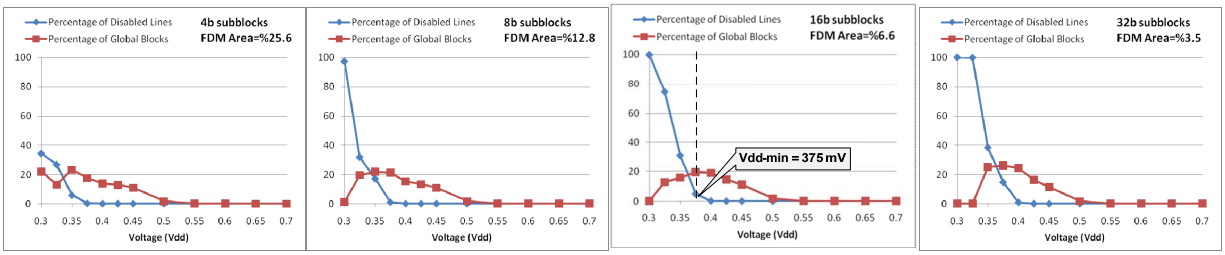


(a) Percentage of disabled lines and global blocks for 4-way L1 (Max global block = 1) while varying V_{dd} and subblock size

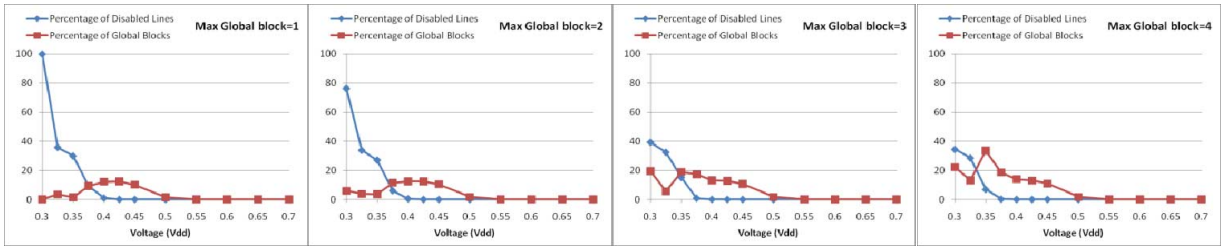


(b) Percentage of disabled lines and global blocks for 8-way L1 (4b subblocks) while varying V_{dd} and Max global block

Figure 9.The effect of changing one design parameter of L1 FFT-Cache while fixing other parameters for different V_{dd} values.



(a) Percentage of disabled lines and global blocks for 8-way L2(Max global block = 4)while varying V_{dd} and subblock size



(b) Percentage of disabled lines and global blocks for 8-way L2(4b subblocks)while varying V_{dd} and Max global block

Figure 10.The effect of changing one design parameter of L2 FFT-Cache while fixing other parameters for different V_{dd} values.

4.3 Design Space Exploration

Figure 9 and Figure 10 present the design space exploration of FFT-Cache for L1 and L2 caches, respectively. We study the impact of various FFT-Cache configuration parameters including subblock size and MGB on the number of target lines/blocks (non-functional 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 non-functional 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 1000 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.

We present the results for different associativity (4 and 8) and various subblock sizes (4, 8, 16 and 32) and various MGB values (1, 2, 3 and 4).

As evident in these figures, decreasing Vdd increases the size of cache non-functional part. It is notable that for very low voltage

(below 400mv), the number of global blocks decreases. Overall, the effective size of cache (cache size – non-functional 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 cache non-functional part.

Increasing MGB, increases the number of non-functional blocks only slightly while it significantly reduces the number of non-functional 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 the FFT-Cache minimum achievable operating voltage (Vdd-min), we limit the size of cache non-functional part and the overhead of the FDM table as described below. The number of target lines/blocks (i.e. the size of non-functional cache part) determines the performance loss of FFT-Cache. To limit the performance loss we assume the relative size of non-functional part to be less than 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 has been highlighted in Figure 9 and 10. Using FFT-Cache scheme the minimum operating voltage for L1 and L2 caches is 375mv. At this voltage the FDM overhead is 7% and 6.6% for L1 and L2 respectively. To reach to such a low voltage level, FFT-Cache has sacrificed only 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 study the impact of FFT-Cache on power and performance.

4.4 Results

A. Performance results

In Figure 11 we report the impact of FFT-Cache on performance of SPEC2K benchmarks. The relative performance degradation in terms of IPC (instruction per cycle) is reported for a 64KB, 4-way set associative L1 FFT-Cache (Figure 11 (a)) and 2MB, 8-way set associative L2 FFT-Cache (Figure 11 (b). We also report the performance degradation when the FFT-Cache scheme is deployed simultaneously in L1 and L2 (Figure 11(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 the majority of 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 baseline 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 benchmark almost no performance loss is reported. These benchmarks include facerec, galgel and lucas. Our investigation indicated that while in these benchmarks the miss rate increased slightly due to cache effective size reduction, the nature of out-of-order execution helped the benchmark to maintain performance.

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

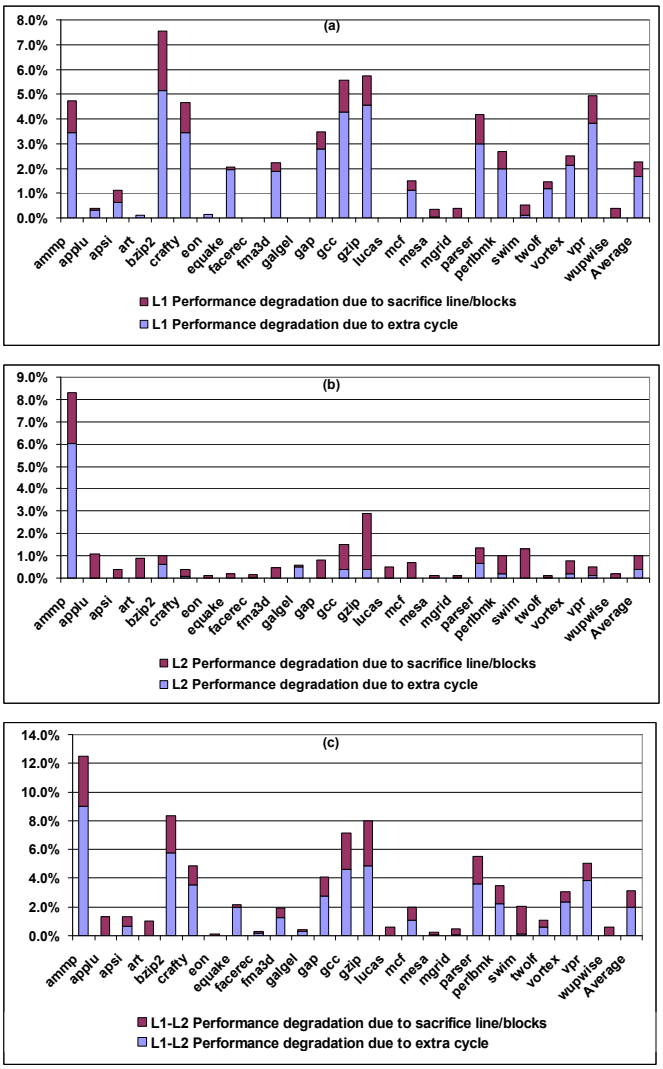


Figure 11.Performance degradation of applying FFT-Cache for (a) L1 (b) L2 and (c) L1 and L2 at the same time.

B. Power/Area Overhead Analysis

Figure 12 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 [10] 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 clock gating and power gating is applied in the FDM array. Therefore, the main source of dynamic power in nominal Vdd relates to bypass MUXs. As it showed in this figure it is less than 3% for L1, but is trivial for L2.

As evident in Figure 12, 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%. Also, 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.

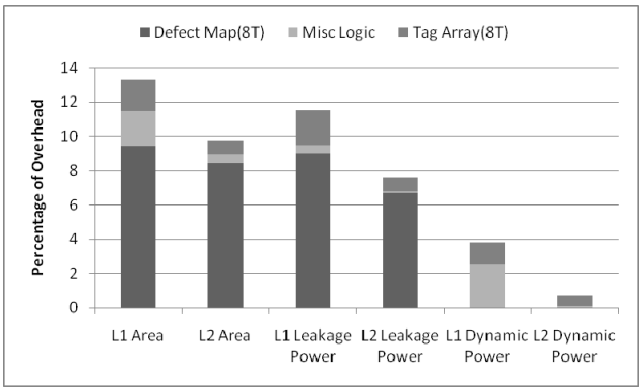


Figure 12.Power and Area overheads of FFT-Cache.

C. Comparison with Recent Methods

In this section we present detailed comparison between our scheme and four state-of-the-art works include Wilkerson et al. [18], ZerehCache [29], 10T SRAM cell [13], and Ansari et al. [30]. Table 2 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.

Table 2.Comparison of different Fault-Tolerant Schemes

Scheme	Vdd-min (mV)	L1 Cache		L2 Cache		Norm. IPC
		Area over. (%)	Power over. (%)	Area over. (%)	Power over. (%)	
6T cell	660	0	0	0	0	1.0
ZerehCache[29]	430	16	15	8	12	0.97
Wilkerson [18]	420	15	60	8	20	0.89
Ansari [30]	420	14	19	5	4	0.95
10T cell [13]	380	66	24	66	24	1.0
FFT-Cache	375	13	16	10	8	0.95

Overall, our proposed FFT-Cache achieves the lowest operating voltage (375mv) and the highest power reduction compared to all other techniques. The closest techniques to ours are 10T cell, Ansari, and Wilkerson. 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 it 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 420mw and thus achieves lower power reduction. Overall, the flexible defect map of FFT-Cache along with high configurability and high flexibility allow it to tolerate higher failure rates compared to other similar techniques.

5. CONCLUSION

In this work, we proposed a fault-tolerant cache architecture, FFT-Cache, which has a flexible defect map 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 can be accomplished by using either block-level or line-level replication in the same set or between two or more sets. It 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 is reduced down to 375mV in 90nm technology. This achieves 66% and 80% dynamic power reduction for L1 and L2 caches, respectively. It also 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.

6. ACKNOWLEDGMENTS

This research was partially supported by NSF Variability Expedition Grant Number CCF-1029783.

7. REFERENCES

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

[2] S. Mukhopadhyay, H. Mahmoodi, and K. Roy. Modeling of failure probability and statistical design of sram array for yield enhancement in nanoscale cmos. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1859–1880, 2005.

[3] W. Wong, C. Koh, et al., "VOSCH: Voltage scaled cache hierarchies," In *Proc. of ICCD*, pages 496-503, 2007.

[4] C. Zhang, F. Vahid, and W. Najjar. A highly configurable cache for low energy embedded systems. *ACM Trans. On Embedded Computer System,* 4(2):363-387, 2005.

[5] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computersystem modeling. *IEEE Transactions on Computers*, 35(2):59–67, Feb. 2002.

[6] M. Guthaus, J. Ringenberg, et al. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE International Workshop on Workload Characterization*, pages 3-14, 2001.

[7] N. Muralimanohar, R. Balasubramonian, and N.P. Jouppi. CACTI 6.5. HP Laboratories, Technical Report, 2009.

[8] L. Chang, D. Fried, et al. Stable sram cell design for the 32 nm node and beyond. In *Proc. Symposium on VLSI Technology*, pages 128–129, June 2005.

[9] G. Chen, D. Blaauw, T. Mudge, D. Sylvester, and N. Kim. Yield-driven near-threshold sram design. In *Proc. of the International Conference on Computer Aided Design (ICCAD)*, pages 660–666, Nov. 2007.

[10] N. Verma and A. Chandrakasan. A 256 kb 65 nm 8t subthreshold sram employing sense-amplifier redundancy. *IEEE Journal of Solid-State Circuits*, 43(1):141–149, Jan. 2008.

[11] F. Moradi, D. Wisland, S. Aunet, H. Mahmoodi, and T. Cao. 65nm sub-threshold 11t-sram for ultra low voltage applications. In *Proc. Intl. Symposium on System-on-a-Chip*, pages 113–118, Sept. 2008.

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

[13] B. Calhoun and A. Chandrakasan. A 256kb sub-threshold sram in 65nm cmos. In *Proc. of IEEE International Solid-State Circuits Conference*, pages 2592–2601, Feb. 2006.

[14] Y. Morita, H. Fujiwara, H. Noguchi, Y. Iguchi, K. Nii, H. Kawaguchi, and M. Yoshimoto. An area-conscious low-voltage-oriented 8t-sram design under dvs environment. *IEEE Symposium on VLSI Circuits*, pages 256–257, June2007.

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

[16] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. C. Hoe. Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding. In *Proc. of the 40th Annual International Symposium on Microarchitecture*, 2007.

[17] P. Shirvani and E. McCluskey. PADded cache: a new fault-tolerance technique for cache memories. In *Proc. 17th IEEE VLSI Test Symposium (VTS)*, 1999.

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

[19] A. Sasan, H. Homayoun, A.M. Eltawil, and F.J. Kurdahi. A fault tolerant cache architecture for sub 500mV operation: resizable data composer cache (RDC-cache). In *Proc. of Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, pages 251-260, 2009.

[20] A. Sasan, H. Homayoun, A.M. Eltawil, and F.J. Kurdahi. Inquisitive Defect Cache: A Means of Combating Manufacturing Induced Process Variation. *IEEE Transactions on VLSI Systems,* 18(12):1-13, Aug. 2010.

[21] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W.Wu, and S.-L. Lu. Improving cache lifetime reliability at ultra-low voltages. In *Proc. of the 42nd Annual International Symposium on Microarchitecture*, 2009.

[22] A. Chakraborty, H. Homayoun, A. Khajeh, N. Dutt, A.M. Eltawil, and F.J. Kurdahi. E < MC2: Less Energy through Multi-Copy Cache. In *Proc. of Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, pages 237-246, 2010.

[23] C. Wilkerson, A.R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S. Lu. Reducing Cache Power with Low-Cost, Multi-Bit Error-Correcting Codes. In *Proc. of the 37th annual international symposium on Computer architecture*, pages 83-93, June 2010.

[24] D. H. Yoon and M. Erez. Memory Mapped ECC: Low-Cost Error Protection for Last Level Caches. In *Proc. of the 36th annual international symposium on Computer architecture,* 2009.

[25] D. H. Yoon and M. Erez. Flexible Cache Error Protection using an ECC FIFO. In *Proc. of the Int'l Conf. High Performance Computing, Networking, Storage, and Analysis (SC'09),* 2009.

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

[27] C. K. Koh, W. F. Wong, Y. Chen, and H. Li. Tolerating process variations in large, set associative caches: The buddy cache. *ACM Trans. on Architecture. and Code Optimization*, 6(2):1–34, Jun 2009.

[28] C. K. Koh, W. F. Wong, Y. Chen, and H. Li. The Salvage Cache: A fault-tolerant cache architecture for next-generation memory technologies. In *Proc. of IEEE international conference on Computer design*, 2009.

[29] A. Ansari, S. Gupta, S. Feng, and S. Mahlke. Zerehcache: Armoring cache architectures in high defect density technologies. In *Proc. of the 42nd Annual International Symposium on Microarchitecture*, 2009.

[30] A. Ansari, S. Feng, S. Gupta, and S. Mahlke. Enabling ultra low voltage system operation by tolerating on-chip cache failures. In *Proc. of the International Symposium on Low Power Electronics and Design*, pages 307-310, Aug 2009.