Maximizing Spare Utilization by Virtually Reorganizing Faulty Cache Lines

Amin Ansari, Shantanu Gupta, Shuguang Feng, and Scott Mahlke, *Member, IEEE*

Abstract—Aggressive technology scaling to 45nm and below introduces serious reliability challenges to the design of microprocessors. Since a large fraction of chip area is devoted to on-chip caches, it is important to protect these SRAM structures against lifetime and manufacture-time failures. Designers typically over-provision caches with additional resources to overcome hard-faults. However, static allocation and binding of redundant spares results in low utilization of the extra resources and ultimately limits the number of defects that can be tolerated. This work re-examines the design of process variation tolerant on-chip caches with a focus on providing the flexibility and dynamic reconfigurability necessary to tolerate large numbers of defects with modest hardware overhead. Our approach, ZerehCache, virtually reorganizes the cache data array using a permutation network to provide more degrees of freedom for spare allocation. A graph coloring algorithm is used to configure the network and identify the proper mapping of replacement elements. We perform an extensive design space exploration of both L1/L2 caches to identify several Pareto optimal ZerehCaches. Given this optimal design points, we employ ZerehCache to extend the effective lifetime of the on-chip caches and prevent early lifetime failures. Finally, yield analysis studies, performed on a population of 1000 chips at the 45nm technology node demonstrated that an L1 design with 16% and an L2 designs with 8% area overheads achieve yields of 99% and 96%, respectively.

Index Terms—Process variation, Wearout, Fault-tolerant cache memories, Manufacturing yield

1 Introduction

As circuit density grows, each transistor gets smaller, hotter, and more fragile. This leads to an overall higher susceptibility of chips to *permanent faults* [10], [36]. These failures can impact the performance guarantees offered by a semiconductor chip, manufacturing yield, and limit their useful lifetime. Efficiency of CMOS technology is questionable in the face of such challenges. Current projections indicate that future microprocessors will be composed of billions of transistors, many of which will be unusable at manufacture time, and many more which will degrade in performance (or even fail) over the expected lifetime of the processor [10]. To address these reliability concerns, designers must equip their designs to tolerate and operate properly in the presence of faults.

On-chip memory arrays in high performance processors are critical for chip reliability as more than 70% of the transistors can be devoted to caches. Therefore, on-chip caches need to be equipped with cost-effective mechanisms to tolerate in-field silicon defects. In this work, we propose a fault-tolerant cache architecture to tackle wearout (over time) and also process variation induced failures (fabrication time). This will extend the effective lifetime of the on-chip caches and prevent early lifetime failures. Assuming no manufacturing defects, Figure 1 depicts the fraction of non-functional SRAM bit-cells for a 2MB L2 cache over time. This plot was generated for a range of mean time to failure (*MTTF*) values from 50 to 200 years. Here, it is notable that even for an MTTF of 200 years, a considerable number of failures need to be addressed in early lifetime. Moreover, a comparison of our scheme with conventional wearout tolerance methods and also the experimental methodology, for generating this plot, will be discussed in section 5.

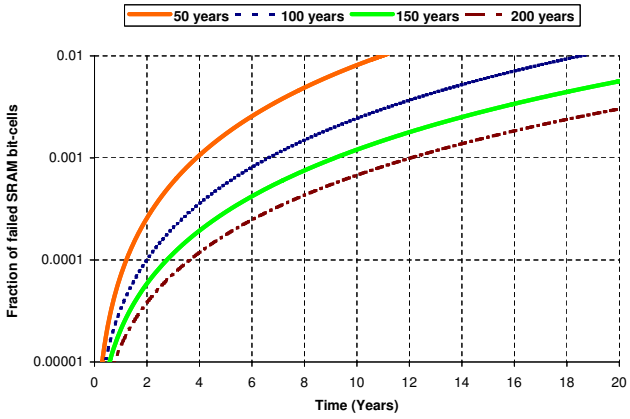


Fig. 1. Fraction of non-functional SRAM bit-cells for a 2MB L2 cache over time. Here, the mean time to failure of each SRAM bit-cell is varied from 50 to 200 years.

Apart from in-field wearout challenges, technological trends into the nanometer regime have lead to an increasing vulnerability of manufactured parts to process variation. A host of factors such as sub-wavelength lithography, line edge roughness, and random dopant fluctuation result in a wide distribution of transistor characteristics which directly translates into lower parametric yield [8]. This significant divergence of process parameters from their nominal specification limits the achievable frequency and also significantly hurts the leakage power of modern high performance processors [38]. SRAM

• *The authors are with the Advanced Computer Architecture Lab, Computer Science and Engineering Department, University of Michigan, 2260 Hayward St., Ann Arbor, MI 48109.*
E-mail: ansary, shangupt, shoe, mahlke@umich.edu.

structures are particularly vulnerable to process variation due to their minimum-geometry transistors, sensitive differential circuit, and area efficient semi-custom layout. Using Berkeley Predictive Technology Model, the yield of an *unprotected* cache in a 45*nm* technology node can be as low as 33%, highlighting the need for appropriate protection schemes [3], [4]. Under process variation, a single SRAM cell can fail because of the following reasons that are sorted based on the frequency of occurrence [3]: **1.** Access Time Failure: It occurs when the differential read voltage between bit-lines is not enough for the sense amplifier to extract the correct stored value. **2.** Write Stability Failure: This case arises when the cell contents cannot be replaced with a new value. This happens due to stronger pull-up of the storage node with input *0* compared to the access transistor. **3.** Read Stability Failure: If the voltage of a storage node with a stored *0* value during the read operation is higher than the trip-point of an inverter that has an output value of 1, the read value from the SRAM cell flips. **4.** Hold Failure: If the supply voltage drops below a minimum level while an SRAM cell is not being accessed, the SRAM cell can lose its stored value.

To illustrate the reliability implications on on-chip caches, Figure 2 presents the probabilities of having at least one faulty SRAM cell considering different granularities of storage. This trend is shown for a wide range of single cell failure probabilities (P_F). Assuming a uniform failure distribution, for a given P_F and a granularity (e.g., byte-level), we performed a Monte Carlo simulation to evaluate in what percentage of simulations, there is at least a single faulty bit-cell. As the failure probability increases in these graphs, the granularity of fault manifestation decreases in size. For instance, as can be seen, the L2 cache configuration demonstrates a modest number of block-level failures at $P_F \sim 10^{-5}$. Thus, a block-level redundancy solution would be satisfactory for fault tolerance in this case. However, at $P_F \sim 10^{-3}$, the L2 cache is certain to contain at least one faulty cell in each cache word-line with a very high chance of fault in each cache block. This makes the use of word-line/block level redundancy impractical. Hence, with the increasing failure probability, a smaller granularity of redundancy will be necessary to guarantee robustness. The same trends hold for L1 cache, but it is favorably shifted towards the right due to the smaller block and word-line sizes of the L1 cache in comparison to the L2 cache. The primary challenge in this scenario is to design a cache architecture that can maintain and optimally utilize the smaller levels of redundancy for defect tolerance. At a 45*nm* technology node, an SRAM cell is expected to have a 30*mV* standard deviation in the threshold voltage (V_{th}) resulting in a P_F as high as 10^{-3} [4]. Thus, there is a real need to devise solutions for these reliability challenges.

To this end, we introduce the ZerehCache (ZC, Zereh in Farsi means body armor), a high-failure rate tolerant solution for both L1 and L2 on-chip caches. ZC is an adaptive, dynamically reconfigurable solution for tackling the high defect rates of future technologies. In order to break the static binding between the spare elements and the original data, ZC virtually swaps the cache word-lines to find the best possible spare allocation. It also provides a wide range of

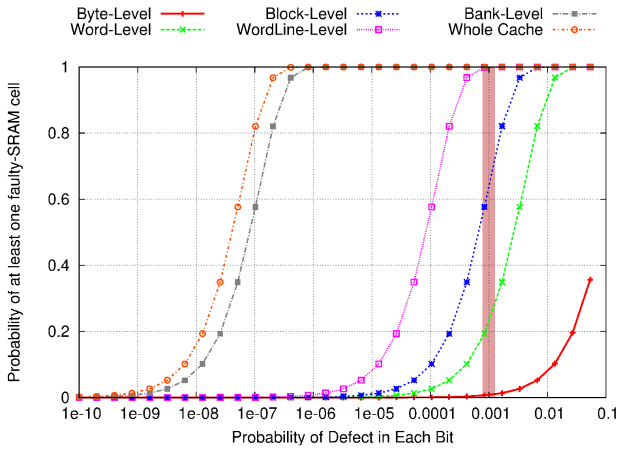


Fig. 2. Probability of having at least one faulty SRAM cell at different granularities while varying the failure probability of each SRAM cell (P_F). This plot was generated for a 2-bank 2MB L2 cache with 128B block size and 256b word size.

cache design options based on the primary design concerns such as delay, power, and area overhead. In this work, the ZC architecture is leveraged to tolerate process variation in 45*nm* technology. ZC takes advantage of its intelligent interlaced usage of redundancies in multiple ways to substantially cut the overheads of protecting on-chip caches. To our knowledge, ZC has the capability of achieving the highest degree of the fault tolerance, among the previously proposed approaches, for a given area budget. Current microprocessors have already been equipped with row-redundancy and other coarse-grained redundancy schemes to protect the caches against hard-faults [32]. ZC can be used as an alternative for these conventional mechanisms while introducing a considerably lower overhead (Section 7). We believe our scheme provides a solid foundation for cache designers to take advantage of the higher clock frequency and transistor density in deeper technology nodes while preserving the correct functionality and timing constraints of their design with less overhead.

The primary contributions of this paper are: 1) A flexible, dynamically reconfigurable architecture that can be leveraged to protect regular SRAM structures against high defect density nanometer technology nodes; 2) Minimizing the amount of redundancy required for protecting the cache by modeling the collision pattern in the main/spare cache with a well studied graph coloring problem and taking advantage of the existing rich approximation methods; 3) A design space exploration in 45*nm* to show the actual process of fixing the architecture parameters; and 4) Evaluating the proposed method under wearout and process variation conditions.

The rest of the paper proceeds as follows. Section 2 discusses related work. In Section 3, we present the architectural details and configuration of ZC. A design space exploration is performed in Section 4 to fix the architectural parameters of our L1 and L2 caches. Given these L1 and L2 ZCs, first, in Section 5, we evaluate the potential benefits of ZC when addressing in-field wearout failures. Furthermore, in Section 6, we evaluate the yield enhancement that can be

achieved using these L1 and L2 ZCs. Section 7 compares ZC against several conventional and recently proposed schemes, and finally Section 8 concludes the paper.

2 Related Work

A significant amount of literature targets the on-chip cache reliability concerns (e.g., transient faults, manufacturing defects, process variation, wearout, and near/sub-threshold operation) and the proposed solutions can be divided into three major categories:

Circuit-Level and VLSI Solutions: Dynamic voltage/frequency scaling can be employed to improve the cache reliability. For this purpose, we need to identify the most vulnerable SRAM cell in each line and scale the access time/voltage to a level that guarantees proper operation for all the cells in that word-line. There are two major drawbacks of this scheme, 1) a mechanism is needed to dynamically determine the weakest cell in each row, and 2) the working conditions of the cache must be adjusted to the weakest cell, resulting in a considerable performance penalty (access latency). A 3T1D DRAM cell can be substituted for the conventional 6T SRAM cell to improve the reliability [25]. However, the 3TD1 cell cannot retain the value for a long period and each word-line must be refreshed periodically. Moreover, since on-chip DRAM is not normally used in current technologies, it adds to the process/manufacturing complexity and effort. Another alternative is to size up the SRAM cells or use a different structure for them (e.g., 8T, 10T, or ST) [23]. Unfortunately, these methods incur a large area overhead (Section 7) and they are mostly employed for power reduction by allowing the near/sub-threshold operation.

Coding Solutions: Simple error detection codes (EDC) and parity can be applied for the detection of the faults in caches [32]. Single error correction double error detection (SECDED) is a widely used technique for protecting the memory structures [14]. However, in a high-failure rate situation, these solutions are not practical because of the strict bound on the number of tolerable faults in each protected data chunk (Section 7). A 2D error correction coding scheme is presented in [19] that uses two sets of EDCs on the rows and columns of the data array. As the results show, this scheme is also not appropriate for high-failure rate situations, like the one addressed in this work. Further, the overhead of updating all the column codes for each cache write is high. Multiple bit error correcting codes (ECCs) like Hamming codes are capable of tolerating high failure rates, but are inefficient in terms of the coding delay, area, and power overheads for on-chip caches [19]. In summary, the coding solutions are best applied to memory structures under low failure-rate scenarios or where transient faults are the main concern.

Architectural Solutions: Dual modular redundancy schemes are used in many designs for providing memory structure reliability, but they are highly inefficient in terms of the overhead [5], [34]. A popular architectural solution is to use redundant rows and/or columns [24]. As seen in Figure 2, the probability of having at least one failure in a row is close to 1.0 for $P_F > 10^{-3}$. This implies that most word-lines can be expected to be faulty from the start, resulting in a poor utilization of the provisioned redundancy. Moreover, since the redundant row replacement is based on a decoder modification and using hard-wired fuses, it is generally not applicable for more than 10 extra rows [16]. A similar set of methods are based on the cache block/row/way disabling that are also suitable for the low-failure rate situations [30]. Wilkerson et. al. have suggested several layers of shifters for merging multiple defective word-lines to form a single functional word-line [42]. To achieve operation in the presence of faults, their Word-Disable method sacrifices half of the cache area and their Bit-Fix method adds three cycles of latency to the cache access time. Both of which result in considerable performance drop-off. There are other works that use a re-mapping table to map a faulty block onto another one [17]. These methods impose a high pressure on the L1-L2 communication bus by increasing the L1 miss rate substantially. Furthermore, these are properly applicable only to the direct-mapped caches [3]. Architectural schemes presented in [31], [21] try to reduce the effective cache capacity and use the adjacent cache blocks to repair faults in one another.

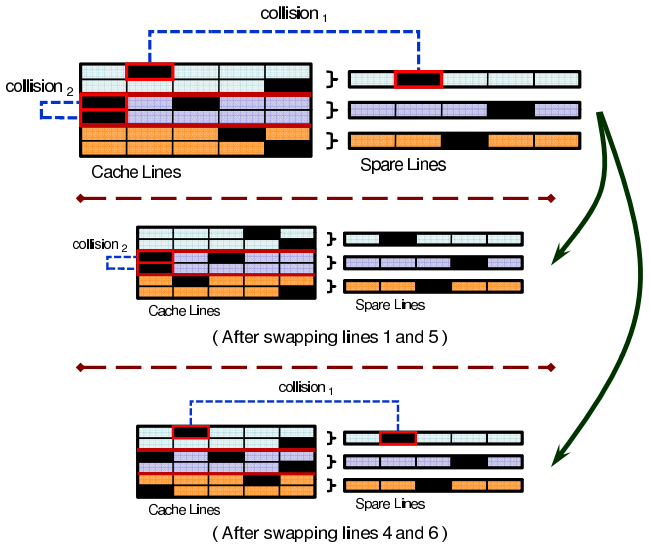


Fig. 3. Two simple scenarios in which the line swapping can preserve the correct functionality of the cache by resolving the occurred collision. A black box shows a faulty chunk of data.

3 ZerehCache

In this section, the ZC architecture is first described that adaptively reconfigures itself to absorb failing SRAM cells. Next, an effective graph-coloring algorithm to configure the underlying architecture is presented.

3.1 ZC Architecture

The key idea behind the ZC architecture is to use redundant units in multiple ways to increase their potential utilization. ZC partitions the complete cache array into sets of equally sized logical groups, where each logical group is allocated one spare cache word-line. Here onwards, we use the term *line*

when referring to a *word-line*. The logical groups are formed by carefully shuffling together physical cache lines in order to optimize the utilization of a single spare cache line. Each cache/spare line is divided up to equally sized data chunks to allow smaller granularities of spare substitution. For instance, the fourth data chunk of the second spare line can be used to substitute the fourth data chunk of the third/fourth cache line in the case of failure. Flexibility for this line shuffling is provided by adding a network into the cache that allows swapping of cache lines to eliminate conflicting failures. Conflicting failures occur when two lines that share a spare line have a failure in the same chunk, or when a cache line and its corresponding spare have failures in the same chunk. While there may be sufficient redundancy, conflicting failures arise and render the cache non-operational.

To illustrate this issue, Figure 3 shows two simple scenarios where ZC can preserve the correct functionality of the underlying cache while it is not possible for conventional redundancy methods to do so. In this figure, each line contains five units of data and every two consecutive lines in the main cache form a logical group. Each logical group is assigned one line in the spare cache. The first line in the main cache has a failure in the same place as the first line of the spare cache. Swapping the first and fifth lines in the main cache can resolve this collision ($collision_1$). After swapping, the second and fifth rows will form the first logical group which utilizes the first row of the spare cache. The second conflict situation, $collision_2$, is between two lines in the same logical group. Swapping the fourth and the sixth lines is one possible way to resolve this conflict. Swapping eliminates collisions and increases the chance of having a functional cache for a given area overhead budget.

A high-level architecture of a set-associative ZC is shown in Figure 4. The cache data array is divided into equal sized groups. Lines within each of these groups share a single spare line in the spare cache (static multiplexing of spares). For each access to the cache array, in a high speed design, the spare cache and the fault map arrays are also accessed in parallel. The result of the fault map access determines whether the spare data chunk should be routed to the output instead of the main cache content. In order to tolerate many defects, logical groups are formed by carefully shuffling together cache lines using an interconnection network. As Figure 3 demonstrates, the functionality of the interconnection network is to swap the lines in a manner that resolves the existing collisions. The configuration for this network is computed once and saved in the non-volatile network configuration storage. By using static multiplexing and the interconnection network for overcoming the limitations of static binding, ZC can maximize the utilization of the spare units. The remainder of this section provides a detailed description for each of the architectural modules.

Spare Cache: Each row in the spare cache corresponds to a logical group of lines in the main cache. A single row of the spare cache is further broken up into smaller redundancy units of fixed size. Each of these redundancy units in the spare cache keeps the valid content of the corresponding corrupted element in the main cache (if any exists). In order to avoid high fan-in ORs required when using the main cache row decoder, the spare cache and fault map arrays use a separate shared decoder. This decoder uses the top *n* most significant bits from the set segment of the memory address, where *n* is based on the number of rows in the spare cache.

Interconnection Network: In order to shuffle around the cache lines and form logical groups, we use an interconnection network. This network is placed between row decoder of the main cache and the cache word-lines. A unidirectional Benes network (BN) [29] is used to provide a non-blocking routing and full permutation mapping between the inputs and outputs. As Figure 4 shows, a BN consists of two back-to-back connected butterfly networks. The main reasons for selecting a BN for this work are: 1) Full permutation and non-blocking properties allow routing any permutation from inputs to outputs in a conflict-free fashion. 2) Logarithmic depth of the net can minimize the imposed delay overhead of the interconnection network. For connecting 2^n nodes to each other, $2n-1$ stages are required. We call this a BN with *n* swapping levels. 3) The BN delay/power/area scaling characteristics are superior in comparison to most other interconnection networks like full-crossbar or omega network.

The network consists of multiple local BNs. Each local BN is used to connect the word-lines with the same relative positions in different groups. For instance in Figure 4, all of the four groups have their 2^{nd} lines connected by a local BN. There have to be as many interleaved local BNs as there are lines in a single group. The set of groups connected by a single local BN is called the swapping set, and the size of this swapping set ($2^{num.\ of\ swapping\ levels} = 2^2$ in the example shown) is determined by the depth of the network. Given the full permutation and non-blocking properties of the chosen interconnection network, lines in the same relative position can be swapped between the different logical groups. Increasing the depth of the BN widens the scope of line swapping, however, it also imposes higher overheads on the underlying cache. In order to minimize these overheads and reach to a network with higher depth, we employ an efficient circuit-level implementation of a BN which is presented in [35]. A memory hash-table can also be used as an alternative here. However, since this network is in the critical path of the cache access, we employ a BN which provides inherent flexibility, lower delay, and lower power consumption.

Network Configuration Storage: The interconnection network configuration is kept in the network configuration storage. According to our evaluations in the next section, a small fraction of the manufacturing test time can be used to solve the configuration and mapping problems. For the network configuration storage, we use a low-voltage *on-chip* NOR-flash described in [37]. However, since this structure is extremely small (mostly less than 400 bytes), employing other non-volatile memories (e.g., fuse or EEPROM) has negligible impact on our results.

Fault Map Array: The fault map array has the same number of rows as the spare cache. For each redundancy unit in a spare cache row, the fault map array stores the row number in the corresponding logical group which utilizes that redundancy. For example, if a broken data chunk in the 5th

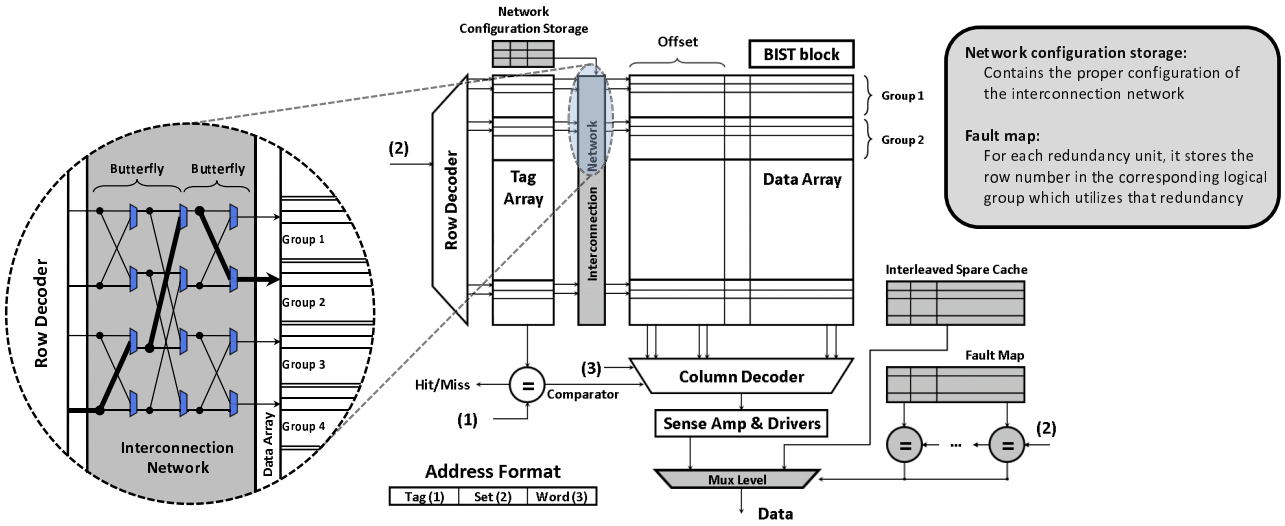


Fig. 4. The high-level architecture of the ZC is shown in this figure and the extra modules that are added to the baseline cache are highlighted. Note that the slices of the base address are shown using numbers 1, 2 and 3 (Address Format). For simplicity, the separate sense amps for the fault map and spare cache, as well as their shared decoder, are not shown. The BIST block is comprised of a Built-in-self-test (BIST) module commonly used for fault diagnosis in the embedded memory structures. Lastly the callout highlights the Benes network which connects the second rows of the four consecutive logical group of rows in the main cache. For illustrative purposes, a single route from the decoder to the word-lines is also shown.

row of an eight-rows logical group should be replaced by its corresponding spare unit, the fault map saves 101 for that redundancy unit. This implies, that for a very small granularity of redundancy, the length of the word-lines in the fault map array can be significantly longer than the main cache. The access time of the fault map array is comparable to the L1 cache. Hence, this structure should be accessed in parallel with the tag array access for the L1. Conversely, for the L2 cache, the access to this structure happens after the hit resolution from the tag side, resulting in a significant reduction in the dynamic access energy. In contrast to the network configuration storage, which should be filled during the manufacturing test time, the fault map gets its contents directly from the built-in self test (BIST) module during the first boot of the system [13]. Further, its content can be saved on the hard-disk and retrieved during the machine boot up. This mechanism works properly for the fault map since the BN routes have already been fixed. And during the testing operation by BIST, the effect of line swapping will be automatically accounted for.

Comparison Stage: This stage compares the least several significant bits of the set segment of the address[1] with the returned content of the fault map array to determine whether that unit of redundancy replaces the data chunk from the main cache.

MUXing Level: At the end of the access critical path, based on the results of the comparison stage, the MUXing level determines for each redundant unit whether the main cache or the spare cache data is valid and drives that onto the cache output. Word-lines in the main/spare cache are divided into equal units of redundancy. The size of these redundancy units specifies the MUXing granularity. Hence, although data chunk size and the MUXing granularity refer to different concepts in the ZC architecture, they are always equal to each other. Moreover, since the read and write are symmetric operations, the only modification in the implementation would be to replace the MUXes with pass transistors.

In order to guarantee the proper operation of the on-chip caches, we assume all the main SRAM structures in the ZC architecture (i.e., main cache, spare cache, tag array, and fault map) would be affected by the process variation. In our design, the main and spare caches are the major contributors to the ZC area and potential failures in these structures are directly handled by our scheme. In order to protect the fault map and tag array, we employ a process variation tolerant 8T SRAM cell which is more area efficient than simple transistor sizing [12], [11], [40]. However, it should be noted that the 8T cell comes with around 36% area overhead and is not cost effective for protecting the entire cache (Section 7).

3.2 Hard-Fault Detection

In this work, we use ZC to tackle process variation as well as wearout induced failures. Here, we separately discuss the mechanisms required to detect these two type of hard-faults. First, we focus on the detection process that is needed to detect process variation induced failures. Since these hard-faults are present at manufacturing time, the standard manufacturing testing process can be employed. In order to test on-chip caches, normally, a combination of automatic test pattern generation (*ATPG*) and the BIST-based testing is used to generate the test vectors, apply them, and produce the compact signature from the test results. Different versions of the MARCH test, introduced by the research community and

1. The number of bits depends on the number of word-lines in each cache logical group.

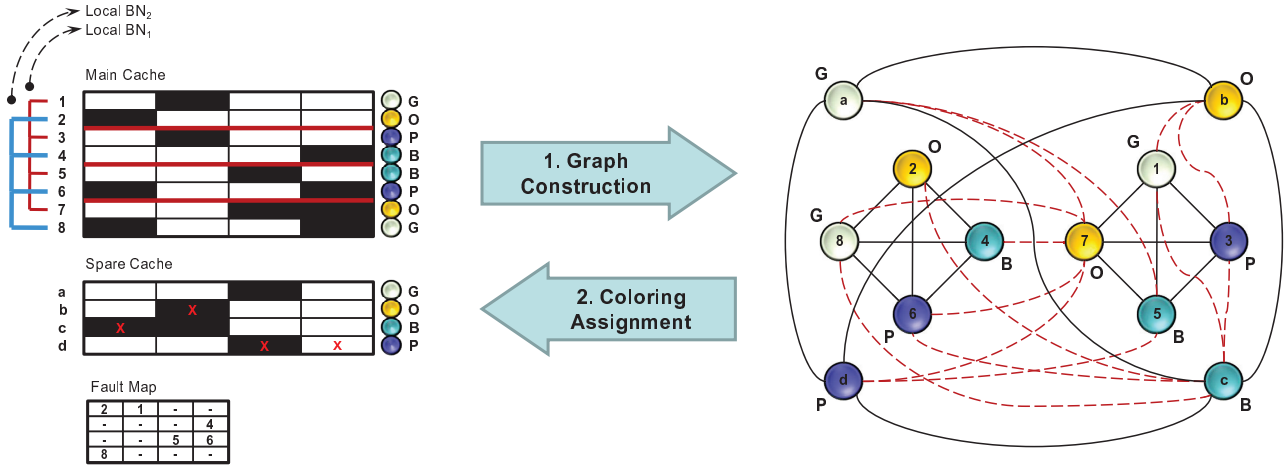


Fig. 5. Mapping between the graph coloring problem and the defect pattern in the main/spare caches. The solid edges stand for the intrinsic conflicts between the word-lines. The dotted edges correspond to the word-line conflicts due to the defect pattern. An "X" indicates a collision using a default grouping. Numbers written in the fault map indicate the corresponding cache word-lines to which the spare units are assigned. (G=Green, B=Blue, P=Purple, O=Orange)

industry, are the most common type of test patterns for testing memory structures and are widely used for testing SRAM structures against stuck-at and bridging faults.

However, wearout induced failures manifest during the lifetime of the system and needs special treatment. Two type of approaches have been proposed for this purpose. The conventional approach is periodic testing in which the system periodically suspends its normal operation and allows the BIST module to test the on-chip caches. In order to guarantee the correct operation of the system, the architectural or microarchitectural state of the system needs to be checkpointed right after each testing interval. Moreover, to reduce the fault detection latency and this checkpointing cost, mostly in terms of main memory or cache usage, this type of testing needs to be done frequently. The other alternative is the continuous testing which is the subject of many current research works. In this type of testing, as soon as a faulty cell gets used, the detection mechanism informs the system of the location of the failure. Continuous testing can have many forms. One of the simplest is error detection codes which are widely used in memory structures. Another well-known proposal is the redundant execution that needs to continuously check the consistency among multiple copies of the same data. Another means for continuous detection is through sensors that can estimate the amount of device level wearout.

3.3 ZC Configuration

Proper configuration of the ZC is crucial for achieving higher utilization of the spare elements. The first step toward this is to determine the input/output mapping for the BNs. In other words, logically group together the cache lines that share a single spare line. We model this as a graph coloring problem that can be solved during the manufacturing test time. The solution to the coloring problem provides the required BN configuration information which is saved in the network configuration storage. On the first boot up of the machine, the BIST module takes advantage of the already configured BNs to find the faulty SRAM cells. The fault map array is then populated by the BIST module based on the location of the faulty cells in the main/spare caches. In order to achieve an effective line swapping capability in ZC architecture, two major algorithmic problems need to be addressed here: 1) Effective group formation, 2) Benes network configuration.

3.3.1 *Effective Group Formation*

The problem of determining the logical groups that share a single spare line in the ZC architecture is modeled as a graph coloring problem. Figure 5 is an example that illustrates the process of mapping the defects in the main/spare cache to a graph. In the cache arrays of Figure 5, each black box stands for a faulty cell. An 8-line cache is divided into 4 logical groups and a spare line is assigned to each logical group. For example, lines 1 and 2 in the main cache form a logical group that utilizes line 'a' in the spare cache. Two local BNs are required to do the proper shuffling. The first (second) lines from different logical groups can swap their positions using the corresponding local BN (e.g., lines 1, 3, 5, and 7 can swap their positions).

The graph on the right hand side of the figure is constructed based on the defect pattern in the main/spare caches. Each node in this graph represents a line in the main/spare cache. Whereas, the edges represent a conflict between a pair of lines, i.e., the two nodes connected by an edge represent lines that cannot be in the same logical group. A graph coloring algorithm can now be applied to this graph to find a solution such that neighboring nodes are not assigned the same color. Thus, after coloring, nodes with the same color are guaranteed to have no edges between them implying that the corresponding cache lines have no conflicts between them. Cache lines with the same color thereby form a logical group. The graph edges that represent conflicts between the lines can be broadly divided into two categories:

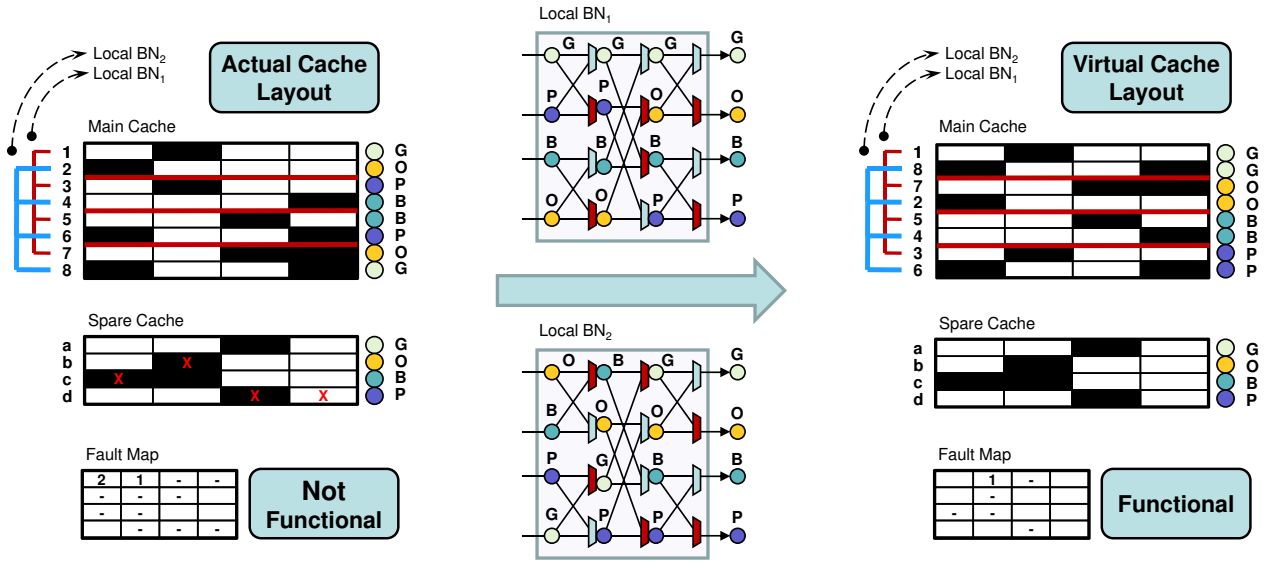


Fig. 6. Proper configuration of two BNs that transform the actual cache layout (left) to the virtual one (right) for the given coloring assignment. The upper (lower) BN connects the first (second) rows of the 4 logical groups. The darker 2-input MUXes are configured to output their lower input while the lighter MUXes output their upper input. (G=Green, B=Blue, P=Purple, O=Orange)

1. Intrinsic Edges: Each of the lines in the spare cache is dedicated to a single logical group in the main cache. This implies that spare lines cannot be in the same logical group. As a result, 4 nodes (a, b, c, d) construct a complete sub-graph (Figure 5). Moreover, the structure of the BN forces the lines connected to a local BN into different logical groups. For example, lines 1, 3, 5, and 7 can not be in the same group. Consequently, these 4 word-lines also form a complete sub-graph.

2. Defect Edges: The defect pattern in the main/spare cache introduces other edges in this graph. These defect edges connect the pair of lines that have at least one conflict (for the same data chunk). For instance, there is a defect edge between the nodes *3* and *c*, because both have their second data chunks faulty.

A graph coloring problem is solvable for a graph G and an integer $K \geq 0$, if the nodes of G can be colored with K colors such that no edge exists between the same colored nodes. For our problem instance, we want to show that if there are h logical groups in the cache, and the nodes can be colored with at most h colors, then there would be a feasible configuration for the BNs such that the ZC works properly. But since there is always a complete sub-graph in the problem graph with h nodes (due to the intrinsic edges), the chromatic number is at least h. On the other hand, our problem constraint dictates that we can use at most h colors for the graph coloring problem. Hence, the graph coloring problem for the ZC configuration should have a solution with exactly h colors. A valid coloring assignment indicates no collision between the lines within each logical group and replacement of the defective data chunks can be properly handled.

Graph coloring is widely recognized as NP-complete. Thus, for solving it, we use an approximate algorithm called Incomplete Backtracking Sequential Coloring (IBSC) [20]. IBSC is a heavily optimized version of the full backtracking solution. It restricts the branching factor on each level to expedite the process of finding the approximate chromatic number. On an average, IBSC only increases the chromatic number of the graph by 5.2%, which is considerably better than the theoretical upper bound that we used for the analysis in Section 4. The complexity of the IBSC algorithm is $O(|V|^4)$ and the actual runtime is discussed in the next section. Furthermore, this algorithm can easily be converted to the exact solution by eliminating the branching heuristic. It is especially useful in the case of small graphs or when more computational power/time can be devoted to the solver.

The graph coloring solution determines the assignment of lines to logical groups. All the lines in the main/spare cache with the same color form a single logical group. For example, all the lines with the orange color are bound to the orange spare row using the corresponding local BNs. Figure 5 illustrates a valid coloring assignment. With this coloring assignment in place, the logical groups formation is complete for the main cache. The next step is to solve the BN configuration problem in order to make the cache functional.

3.3.2 *Benes Network Configuration*

The BN is non-blocking and also allows any permutation of the inputs to be mapped to the outputs. In Figure 6, the left cache structure shows the physical cache layout with a solution for the graph coloring problem. As described, the color of each line determines the logical group to which the line is assigned. As a result, the position of each particular line after the line swapping is apparent. For instance, in the Figure 6, the last cache row has a green color which corresponds to the first row of the spare cache. This denotes that the last cache row should be mapped to the second row of the first logical group. The line ordering for the virtual cache layout on the

right hand side can be obtained from the physical layout by employing two 3-layer deep local BNs. The BNs have to be properly configured, by determining the select signals for the MUXes within the BN, to achieve this re-ordering. Having the desirable permutation between the inputs/outputs of the BNs, we employ the recursive method described in [43] to configure the network. Since an n-input BN is constructed from two identical $\frac{n}{2}$-input sub-networks, the configuration can be computed recursively in $O(n^2)$.

4 Design Space Exploration

The process of finding suitable design points for L1/L2 ZCs involves fixing the architectural parameters. The high level architectural parameters used for this exploration are listed in Table 1. In addition, there are three main parameters specific to the ZC design: a) size of the spare cache, b) depth of the BN, and c) the MUXing granularity for the redundant data chunks. In this section, we sweep a wide range of values for these parameters and study the overhead of each design point. The number of spare cache lines was taken from the set $\{2^i \mid i \in \{0, 1, ..., 7\}\}$. Note that the length of the word-lines is the same for the main and spare caches. The depth of the BN is selected from the set $\{1, 3, 5, ..., 19\}$ and the MUXing granularity is selected from the set $\{2^i \mid i \in \{0, 1, ..., 10\}\}$ bits. In total, considering both the L1 and L2 caches, there are 1760 points in the design space. In order to prune this design space, a number of practical design constraints were considered. For instance, designs with more than 128 spare lines were not studied due to their significantly high area/power/delay overhead. Detailed discussions of these practical constraints and their impact on the design space follows. Finally, we pick suitable configurations for L1/L2 ZCs.

1) Graph Coloring Solver Time: A deeper BN can provide a wider range of cache line swapping, thereby improving ZC defect tolerance. However, a deeper network also increases the complexity of graph coloring and BN configuration. Out of these two, the runtime of the graph coloring problem is by far the dominating factor. Figure 7 depicts the relationship between the size of the graph-coloring problem and the time required to solve it using the IBSC algorithm (Section 3.3).

TABLE 1
The target system configuration

Parameters	Value
Frequency	4 GHz
L1 Caches	64KB data and 64KB instruction, 2-way, 64B word-lines, 2 cycles hit latency, 32B block size
L2 Cache	2 banks 2MB Unified, 16-way, 1KB word-lines 12 cycles hit latency, 128B block size
Registers	128 integer, 128 floating point
ROB (re-ordering buffer)	128 entries
LSQ (load/store queue)	64 entries
Instruction fetch buffer	32 instructions
Issue width	4
FU (functional unit)	4 int ALU, 1 int mult/div, 2 memory system ports
FPU (floating point unit)	4 FP ALU, 1 FP mult/div
Main memory	250 cycle latency, 16 bytes with 10-cycle latency
Branch predictor	combined (bimodal and 2-level)
BHT (branch history table)	4096 entries
RAS (return address stack)	32 entries
BTB (branch target buffer)	512 entries, 8-way associative

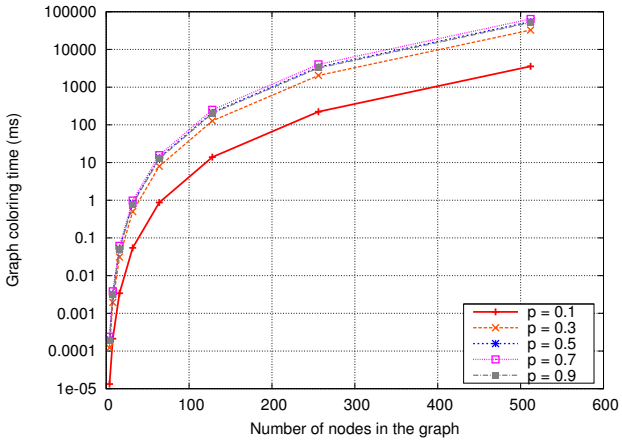


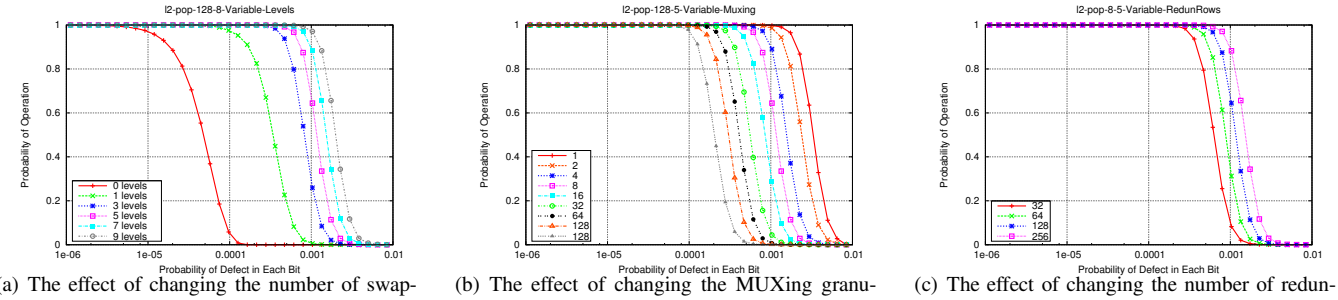
Fig. 7. The run-time of the IBSC graph coloring solver in *ms* for different edge densities and number of nodes in the graph. In this figure, *p* is the edge density which is defined as the probability of having an edge between an arbitrary pair of nodes in a random graph *G(n,p)*.

We ran the solver on a single core Pentium-4 processor with $1GB$ of memory capacity and $2GHz$ clock rate. The Y-axis in this plot is logarithmic. It demonstrates the fast growth in the runtime of the solver with the increase in the problem size.

The total manufacturing test time for a high-end processor (considering the functional, structural, wafer, and packaging tests) is around a few minutes [39]. Using this as a reference, we limit the graph solver time to use a maximum of 10 seconds for all four on-chip cache structures (L1-D, L1-I and two banks of L2). For the case when the cache has 1 spare word-line for every k word-lines and the depth of the BN is $2b - 1$, the total number of nodes in the graph coloring problem would be $(k + 1)2^b$. According to Figure 7 and based on our solver time budget, ZCs can use BNs that are up to 9 levels deep and can connect 32 logical groups together. For instance, if each logical group consists of eight word-lines, there would be $32 \times (8 + 1) = 288$ nodes in the graph coloring problem.

There is a less than 4% chance that the solver does not find a feasible coloring assignment due to limitations on the time budget or the inherent complexity of the collision pattern (Section 6). Using a deeper BN, longer time budget for the solver, finer granularity of MUXing, or a larger spare cache can further reduce this small chance. Nevertheless, if such a situation does arise, we can either resize the cache or simply reject it. Block/way disabling techniques [30] can also be applied at the position of the faulty cell to preserve correct functionality. Note that the scenarios where ZCs need to resort to such methods are very rare.

2) Probability of Operation : The probability of operation (P_{op}) is a definitive metric for a reliable system. We calculate the probability that a specific ZC architecture can properly operate for a given P_F and use the results to further prune our design space. The graph, which was generated in Section 3.3, represents an instance of a defective cache. For the sake of this study, defective caches are modeled as random graphs *G(n,p)* since SRAM cell defects occur as random events. These random defects are due to the major contribution of the random dopant fluctuation to the process variation [4]. Here, *n* is the



(a) The effect of changing the number of swapping levels while using 128 spare word-lines and MUXing granularity of 8 bits.

(b) The effect of changing the MUXing granularity while using 128 spare word-lines and five levels of swapping.

(c) The effect of changing the number of redundant rows while using MUXing granularity of 8 bits and five levels of swapping.

Fig. 8. P_{op} of L2 ZC for different P_F while fixing two parameters and allowing the third one to vary.

number of nodes and p is the probability of having an edge between an arbitrary pair of nodes.

The next step is to estimate the graph coloring solution for these graphs. Calculating the average upper bound of the chromatic number for a random graph is a challenging problem in graph theory [7]. We use two different proposed upper bounds to evaluate the P_{op} of ZCs for a given number of failures. The same set of input conditions are required for both of these upper bounds, which were derived by Achlioptas [1], [2] and Bollobas [9]. The proposed upper bound by Bollobas (B) works better for smaller values of p and the Achlipotas bound (A) is mostly applicable for the larger values of p. Thus, we used the weighted average of these two bounds based on the p value (e.g., $pA + (1-p)B$). Approximation algorithms used to derive these upper bounds, have a significantly poorer approximation factor compared to the IBSC algorithm used in Section 3.3 [26], [41]. The edge probability factor p is defined as the ratio of the expected number of edges in the graph to the number of edges in K_n (a complete graph with n nodes). The expected number of edges in a randomly constructed graph can be calculated by accounting for the intrinsic and fault edges. In other words, $E_{edges} = E_{Intrinsic} + E_{Fault}$ where:

$$E_{Intrinsic} = m\binom{u}{2} + \binom{u}{2} \quad \text{and}$$

$$E_{Fault} = m \times u^2 \times \left[1-(1-\alpha_1\alpha_2)^t\right] + \left[\binom{u\times m}{2} - m\binom{u}{2}\right] \times \left[1-(1-\alpha_2^2)^t\right]$$

Here, m is the number of word-lines in each logical group, u is the number of logical groups in a swapping set, b is the MUXing granularity, t is the number of redundancy units in each word-line, n is number of swapping sets, p_1 is P_F for the main cache, and p_2 is P_F for the spare cache. Here, $\alpha_i = 1-(1-p_i)^b$ shows the probability of having at least one failure in b bits.

Figure 8 shows the P_{op} of an L2 ZC with 128 spare word-lines, MUXing granularity of 8 bits, and 5 levels of swapping. In each of the sub-figures, two of the parameters are fixed and the third one gets the values from the original sweeping set. Notice that in Figure 8(a), adding the first few levels of line swapping significantly increases the robustness of the cache, but beyond 3 levels, adding more levels has a diminishing return. Since the weighted average of the two bounds is not an integer number, we employed a semi-sigmoid function, which is a sigmoid function fitted to the shifted step function (*Number of Logical Groups + 0.5*), for mapping the calculated chromatic number to P_{op}. Using this semi-sigmoid function, if the calculated chromatic number is smaller than the number of available logical groups in the swapping set, the graph is colorable with a probability close to one and if the derived chromatic number is one unit larger than the number of logical groups, the probability would be close to 0. As shown in [3], P_F in 45*nm* can be as high as 10^{-3}. Based on this fact, we pick the design points from our design space that have $P_{op} > 90\%$ for $P_F = 10^{-3}$.

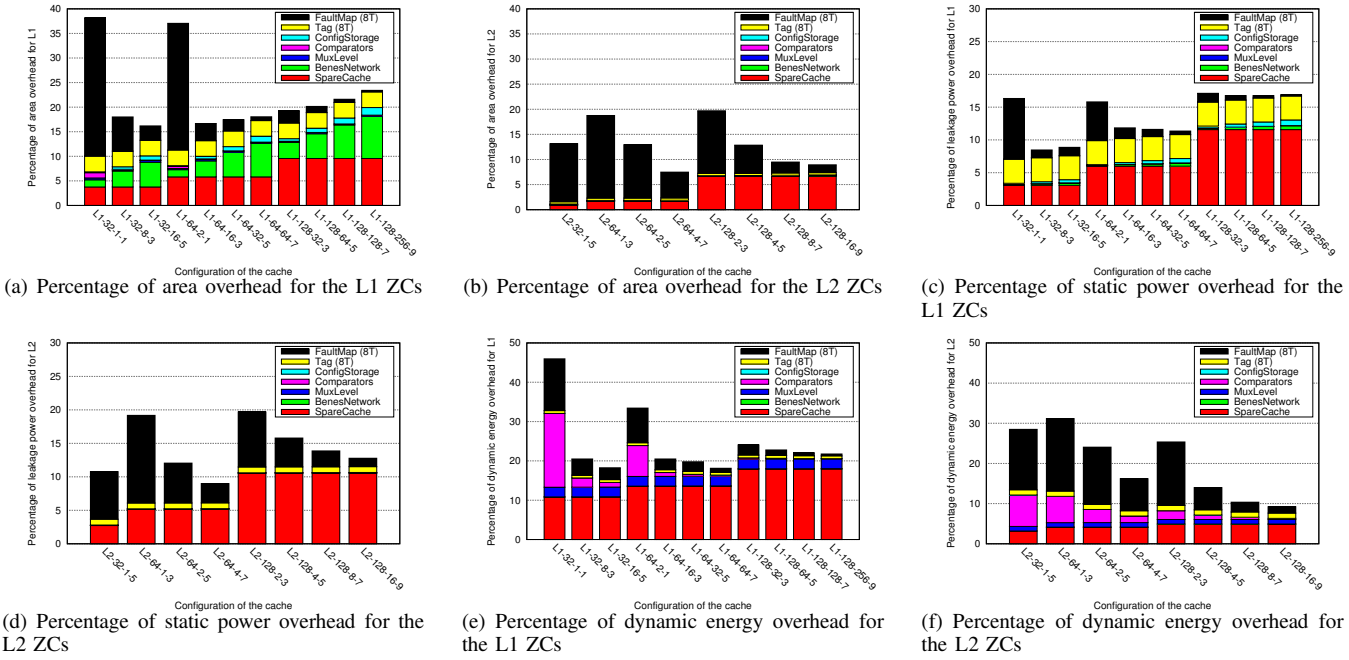
3) Area and Power Overheads: Based on the limiting factors that we have proposed, the size of the design space shrinks from the 1760 starting points down to 103 points. The next factor for eliminating the points is a one-by-one comparison. Given a design point (L_1, M_1, D_1) with L_1 spare word-lines, MUXing granularity of M_1, and a D_1 deep BN and another design point (L_2, M_2, D_2), we can exclude the first point from the design space if it is inferior in all dimensions:

$$L_1 \geq L_2 \ , \ M_1 \leq M_2 \ , \ D_1 \geq D_2$$

This is equivalent to removing dominated points in the Pareto space with dimensions L_1, M_1, D_1. This step reduces the design space to 11 points for L1 and 8 points for L2.

To evaluate our designs, we used CACTI 6.0 [28] for evaluating the area, leakage power, and the dynamic energy for the SRAM structures. Furthermore, we created structural Verilog models for the non-SRAM structures which are introduced by ZC (e.g., MUXing level and BNs). Synopsys Design Compiler was employed to evaluate the area and delay of these structures while Synopsys Power Compiler was used for evaluating the dynamic and static power. All designs are evaluated in 45*nm*.

Figure 9 shows the area, leakage power, and dynamic energy overhead of the selected points in the design space. For instance, L1-32-8-3 stands for an L1 design point with 32 redundant rows, MUXing granularity of 8 bits, and BNs of depth 3. It is notable that increasing the size of the spare cache does not always lead to an increase in the area of the L2 ZC because the fault map size is reduced. However, due to the longer L2 word-line, a finer MUXing resolution is required which results in a relatively larger fault map array for L2



(a) Percentage of area overhead for the L1 ZCs (b) Percentage of area overhead for the L2 ZCs (c) Percentage of static power overhead for the L1 ZCs

(d) Percentage of static power overhead for the L2 ZCs (e) Percentage of dynamic energy overhead for the L1 ZCs (f) Percentage of dynamic energy overhead for the L2 ZCs

Fig. 9. Area, power, and energy overhead of the potential L1/L2 ZCs which are stated in percentage.

compared to L1. The dynamic energy overhead for the L1 ZC was mostly higher compared to the L2 ZC. There are two reason behind this: 1) The L1 cache accesses the fault map/spare cache in parallel to the main cache and 2) The L1 cache reads the entire set for every access whereas the L2 cache is able to read just the right cache block because the tag and data are accessed sequentially.

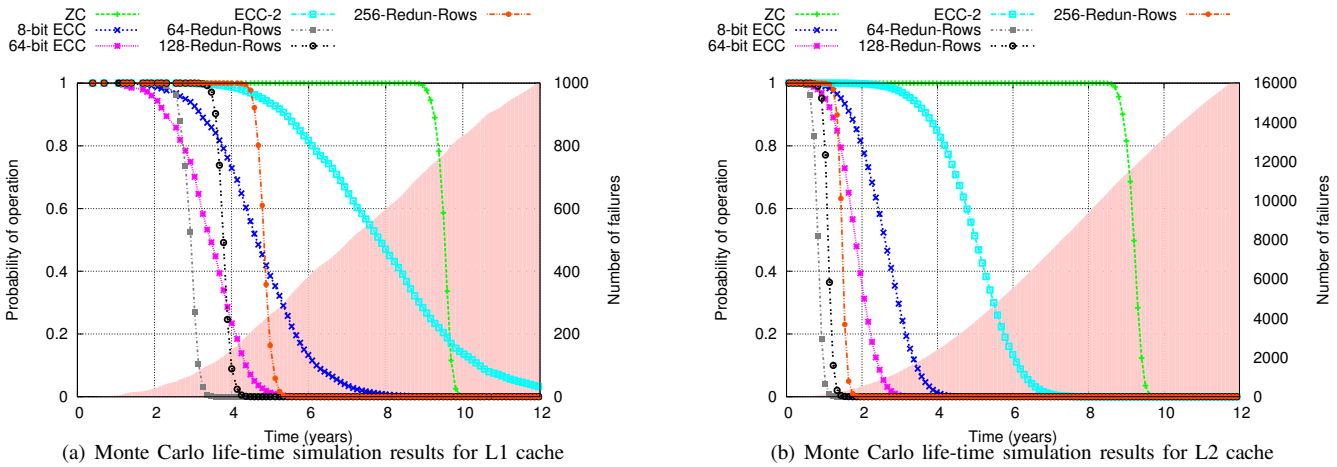
4) Cache Access Latency: The increase in the BN depth also has a direct impact on the cache access time. Since on-chip caches are essential for the performance of modern processors, we assume no slack is available on the access time of the caches. Therefore, any minor modification in the base caches results in at least one extra cycle access penalty. Nonetheless, in the case that considerable slack is available, a design with narrow BN can be leveraged for avoiding any additional cycle latency. In our design, the MUXing level and BN are on the critical path of cache accesses. Based on the timing analysis of our design, in Figure 9, design points with BN depth less than or equal to 7 need one extra cycle latency for the cache access while others (i.e., BN depth = 9) require 2 extra cycles. In Section 7, we evaluate the performance drop-off due to the additional access latency of the L1/L2 ZCs.

Considering the design points in Figure 9, we select L1-32-16-5 as the L1 ZC which imposes 16% area, 9% static power, and 19% dynamic energy overhead over the baseline L1 cache. For the L2 ZC, L2-64-4-7 is selected which imposes 8% area, 9% static power, and 16% dynamic energy overhead compared to the baseline L2 cache. These two selected configurations represent a good trade-off between all the design objectives. However, based on a particular optimization criteria, another design point might work better. For instance, if static power is the main concern, the optimal design point for L1 switches to L1-32-8-3.

5 Wearout Tolerance

In the last section, we fixed the architectural parameters of our L1 and L2 ZCs. Here, we evaluate the potential benefits of our proposed cache architecture when addressing wearout-induced failures. Wearout, in contrast to process variation, is a gradual process. Supposing that the lifetime of a cache can be extended to 10 years and it will experience twenty thousand cell failures during this period, the mean time between failure (MTBF) would be 43.8 hours – worst case scenario. Given the fact that the required time for solving the graph coloring and BN configuration problems is only several seconds, our method can be easily applied to reconfigure the cache after detection of each failure. As another notable point, on-chip flash could also be replaced with latches in this case since the configuration problems would not need to be solved during the manufacturing test time. Nonetheless, even if on-chip flash was deployed, the write-cycle limitations of a flash cell ($\sim$100K) is still higher than the maximum number of required reconfigurations. As a result, it allows us to reconfigure the BN more than 16000 times – that is the average upper-bound for the number of tolerable faults based on the L1/L2 ZCs selected in Section 4. We measured the area/power overhead of replacing on-chip flash with latches. Although negligible for L2 cache, the area overhead of the BN configuration storage for L1 is still noticeable and increases the overhead of our selected L1 ZC from 16% to 18%. Moreover, in this scenario, the configuration information could be stored in the hard-disk after it is obtained. When a system is powered up this information is retrieved from the hard-disk and moved to the network configuration storage.

Upon the detection of a new fault, since that is the only outstanding faulty cell, block/way [30] disabling techniques can be applied based on the position of the faulty cell to pre-



(a) Monte Carlo life-time simulation results for L1 cache

(b) Monte Carlo life-time simulation results for L2 cache

Fig. 10. Results of Monte Carlo lifetime simulation which show the probability of operation for L1/L2 caches protected by different mechanisms. In addition, the shaded region shows the expected number of failures over the life-time.

serve correct functionality of the underlying cache. Removing this faulty cell from the functional space of the cache enables ZC to use the rest of the available cache space to solve the configuration problems. A simple cache disabling mechanism is to only use the half part of the cache which does not contain the faulty cell and disable the other half. After ZC reconfiguration is performed, the whole cache space would be functional.

A Monte Carlo engine is employed to study the P_{op} for ZCs over their life-time. In each iteration of the Monte Carlo simulation, time to failure (TTF) for each SRAM cell in various array structures is calculated using a Weibull distribution with a nominal mean of 100 years – as the expected lifetime of an individual cell [36]. Hundreds of such iterations are run during the entire simulation. The simulation results are shown in Figure 10 for the L1/L2 ZCs selected in Section 4 and several other conventional protection mechanisms. In this figure, SECDED is employed for implementing 8-bit and 64-bit ECCs.

One advantage of ZC over the conventional protection mechanisms is its ability to equalize the lifetimes of L1 and L2 caches. This implies the proper relative provisioning of the caches against hard faults. As a result, ZCs maximize the utilization of the entire provisioned spare elements. The shaded region in these figures depicts the cumulative distribution function (CDF) of the combined MTTF Weibull distributions for the main/spare caches. For instance, in Figure 10(a), there would be ∼400 faulty cells in the L1 related SRAM structures after 6 years. As can be seen, the selected ZC architectures prolong the functional lifetime of the caches up to 10 years. Furthermore, P_{op} has a graceful degradation for the ECC methods compared to the sharp drop-off for ZC and row/column redundancy. Consequently, there is a significant chance for the ECC protected cache to break early in the life-time. This makes them an inappropriate choice even when a long life is not expected. Two bit correction ECC (DECTED), on the other hand, needs 14 extra bits for each 64-bit of data which is ∼22% overhead only for keeping the error correction bits. In terms of the energy overhead imposed by the encoder/decoder per access, as shown in [18], around 50% should be expected.

6 Yield Analysis

In this section, we go through the process of manufacturing yield calculation for a population of ZC enabled chips. A population of 1000 chips was generated from the selected ZC configurations for this purpose. We account for both, inter-die (die to die (D2D)) and intra-die (within die (WID)), components of the process variation. VARIUS [33] is leveraged to model systematic, D2D, and module level intra-die variations. Each chip is considered as a composition of 8 SRAM structures: L1-Data, L1-Inst, two L2 banks, and the corresponding spare caches. The $\frac{\delta_{V_{th}}}{V_{th}}$ is set to 12.5% which is the projected Systematic + D2D variation for 45*nm* technology [4].

Having all the high level variation models in place, a two-step approach is used to derive the number of faulty cells in each SRAM array for an arbitrary chip in the population: 1) We take the intra-module variation model from [4] with $\delta_{V_{th}} = 30mV$. Using this model, the nominal value of P_F across each module is derived from the data provided in [27] based on the average shift in V_{th} for that module. 2) The clustering effect, which determines the degree of defect dispersal in the cache structures, is also modeled. Due to the high density of SRAM structures, the clustering effect has a significant impact on the arrangement of the defects in the corresponding SRAM arrays. We account for it by employing the large-area clustering negative binomial model [22] which is based on the well-known negative binomial yield formula.

Figure 11 illustrates the distributions of the 1000 generated chips based on the number of faulty SRAM cells in their L2 caches. For instance, as Figure 11 shows, around 40 of the chips (∼4%) have 4185 to 5021 faulty SRAM cells in their L2 cache. These derived distributions are consistent with the ones in [3]. It is interesting to note that, in the case with no protection scheme for the cache, the yield of a 2MB L2 in 45*nm* technology could be as low as 33%. We conducted the

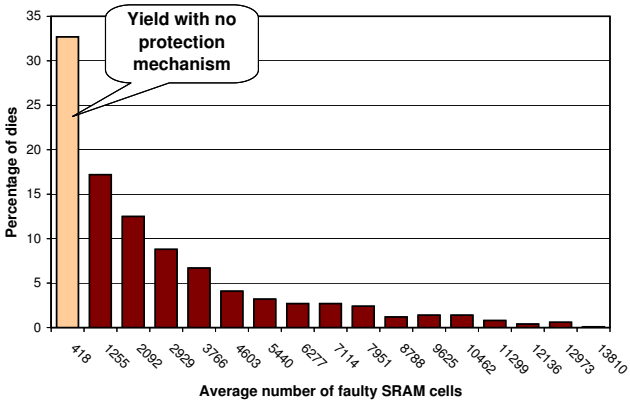


Fig. 11. Distribution of generated chips by the number of faulty SRAM cells in their L2 caches. A population of 1000 chips is generated by considering the large-area clustering effect, intra-die, inter-die, systematic, and parametric variations.

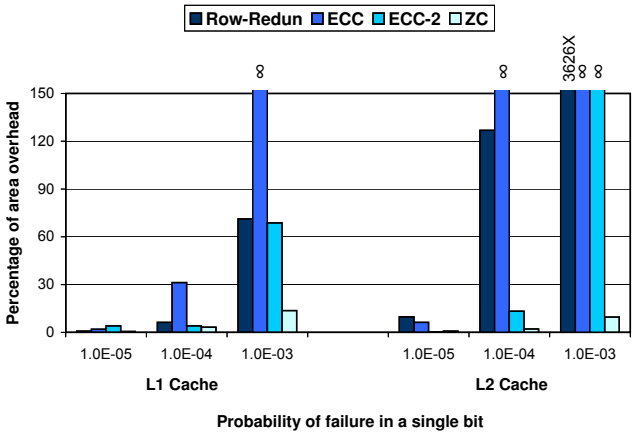


Fig. 12. Area overhead of the different protection mechanisms for tolerating a given P_F. In this figure, Row-Redun stands for the row redundancy protection scheme. ECC and ECC-2 are the 1-bit and 2-bit error correction schemes, respectively.

same set of experiments for our 64KB L1 cache. Likewise, the yield of the baseline L1 cache with no protection mechanism can be as low as 36%.

Manufacturing yield is defined as the fraction of fully functional chips to the total number of manufactured ones. This value can be interpreted as the probability of operation for a particular chip after the manufacturing process (Y_{chip}). We define *CW* and C_i as events that express the proper functionality of a manufactured chip and the existence of *i* faulty cells in a chip, respectively. In the following equations, N_{tot} is the total number of manufactured chips, N_i is the number of the chips with *i* faulty cells, and N_{cells} is the total number of SRAM cells. Based on the rules of probability:

$$\begin{aligned} Pr(CW) &= \sum_{i=0}^{N_{cells}} Pr(CW \cap C_i) \\ &= \sum_{i=0}^{N_{cells}} Pr(CW|C_i) \times Pr(C_i) \\ &= \frac{1}{N_{tot}} \sum_{i=0}^{N_{cells}} Pr(CW|C_i) \times N_i \end{aligned}$$

Since we consider an independence between the P_F of L1 and L2 caches, as shown in [17], the yield of a chip can be written as:

$$Yield_{chip} = \prod_{i \in chip\ modules} Yield_i \qquad (1)$$

As a result, $Pr(CW)$ can be written for each cache separately. Equation 1 is used to calculate the chip yield in each case. Here, $Pr(CW|C_i)$ is the probability of having a functional cache given that it contains *i* faulty cells and, using probability chain rule, can be written as:

$$\begin{aligned} Pr(T, FM, MC, SC|T_{i_1}, FM_{i_2}, MC_{i_3}, SC_{i_4}) = Pr(T|T_{i_1}) \\ \times Pr(FM|FM_{i_2}) \times Pr(MC, SC|T, FM, MC_{i_3}, SC_{i_4}) \end{aligned}$$

where $i_1 + i_2 + i_3 + i_4 = i$. In this equation, *T/FM/MC/SC* are the events that the tag/fault map/main cache/spare cache arrays work properly. Similar to the previous equations, T_{i_1} is the event of having i_1 faulty cells in the tag array. For the fault map and tag array, we assume 8T cells guarantee the fault-free operation of these relatively small structures (i.e., $Pr(FM|FM_{i_2}) = 1$ and $Pr(T|T_{i_1}) = 1$). Since events F and T are always true, we assumed independence between these two events in our derivation. Finally, calculation of the last term is discussed in Section 4.

Given the population of 1000 (N_{tot}) generated chips, N_i for each of the cache structures is known using the mentioned modeling. Yields of the L1 cache and each L2 bank are calculated through the described methodology. The derived yield for the L1 ZC and each bank of L2 ZC are 98.8% and 98.2%, respectively. This implies 96.4% yield for the L2 ZC.

7 Comparison and Discussion

To demonstrate the efficiency of our design, we compare ZC with conventional and recently proposed methods in this section. As representatives for the ZC architecture, we pick the L1-32-16-5 configuration as the L1 ZC (16% area overhead and 99% yield) and L2-64-4-7 configuration as the L2 ZC (8% area overhead and 96% yield).

7.1 Comparison with Conventional Techniques

Figure 12 demonstrates the amount of area overhead required to protect the L1/L2 caches using different protection schemes. For a given probability of failure, we started with the least possible overhead for every mechanism and gradually increased the area overhead until the P_{op} reaches 90%. An infinity symbol (∞) on the top of a bar indicates that achieving $P_{op} > 90\%$ is not possible for the corresponding protection mechanism. This figure only accounts for the amount of *redundancy* required by SECDED (ECC), DECTED (ECC-2), and row-redundancy methods while considering the complete overheads for ZC modifications. In other words, hardware

TABLE 2
Comparison with recently proposed cache protection schemes

Protection scheme	L1 Cache			L2 Cache			Norm. IPC (SPEC-2K)
	Area over. (%)	Disabled (%)	Power over. (%)	Area over. (%)	Disabled (%)	Power over. (%)	
Wilkerson [42]	15	50	61	7	25	27	0.89
8T [12], [11], [40]	36	0	16	36	0	22	1.0
ZerehCache	16	0	15	8	0	12	0.97

overhead for encoder/decoder is not considered for ECC/ECC-2. Similarly, the decoder augmentation is not included in the area overhead of the row-redundancy protection method.

Row-redundancy can protect any cache with inefficient usage of the redundant elements. Nevertheless, as it is shown in [16], row-redundancy with more than 10 extra rows is not efficient due to the considerable increase in the row decoder latency. As shown in this figure, the area overhead of ZC is significantly smaller compared to even the 2 bit error correction scheme (ECC-2) which has a significant power and area overhead for decoding/encoding. Going beyond 2 bit correction using ECC codes is extremely expensive in terms of the code storage area, decoding/encoding power and delay [19]. On the other hand, single bit correction ECC cannot even protect the cache structures with $P_F > 10^{-4}$. For L2, the difference between ZC and other protection mechanisms is even more noticeable because of the longer word-line and larger cache size that challenge the other protection mechanisms. In terms of the energy consumption, ECC and ECC-2 impose around 25% and 50% overheads, respectively [19]. Whereas, both of the selected L1/L2 ZCs have less than 20% energy overhead (Section 4). Hence, it should be clear that the conventional soft-error cache protection schemes cannot deal with the high degree of process-variation in deep nanometer technologies.

7.2 Comparison with Recently Proposed Techniques

More recent proposals target high defect density scenarios that are challenging if not impossible for conventional schemes. Here, we compare ZC with three of these recently proposed cache reliability schemes that target failure rates close to ours. For the purpose of comparison, we measure the performance drop-off for a system (Table 1) equipped with the selected L1/L2 ZCs in Section 4. A performance loss is expected due to the extra cycle of latency added to both L1 and L2 ZC designs. We used the SimpleScalar [6] out-of-order simulator along with the SPEC-FP-2000 (171.swim, 172.mgrid, 173.applu, 177.mesa, 179.art, 183.equake, 188.ammp) and the SPEC-INT-2000 (164.gzip, 175.vpr, 176.gcc, 181.mcf, 197.parser, 255.vortex, 256.bzip2) benchmarks. On average, a 3.2% performance drop-off is observed, with maximum of 6.9% for 197.parser and minimum of 0.1% for 176.gcc.

Agarwal [4] proposed a fault-tolerant *direct-mapped L1* cache that uses cache block remapping. This method maps faulty blocks to the neighboring functional blocks in the same word-line, which forces the L1 to access L2 for getting the values of these blocks. This method is only applicable to direct-mapped caches and cannot be efficiently applied to L2. As shown in Figure 1, around 64% of the L2 cache blocks are faulty and the value of these blocks must be retrieved from the main memory. For our system configuration (Table 1), this results in an effective access time of 164 cycles for L2 which hurts the performance drastically. Nevertheless, considering only L1, they achieved 94% yield compared to 99% yield for our scheme.

Wilkerson *et. al* [42] proposed two cache protection schemes that use several layers of shifting to merge multiple defective lines into a single functional line. Their method was originally designed to reduce the operational voltage by tolerating unwanted SRAM failures. Alternatively, in order to improve the stability of an SRAM cell, Chang *et. al* [11] proposed an 8T SRAM cell, which has been studied and compared with the other alternatives in a more detailed manner by Chen [12] and Verma [40]. These works show that 8T is more effective than simple transistor up-sizing for improving the stability of a bit-cell. An 8T cell is more robust against read upset failures compared to a conventional 6T cell due to the isolation of the read and write paths [12].

Table 2 summarizes the comparison with these two schemes. As can be seen, Wilkerson's method has a notably higher performance drop-off than ZC. This behavior is due to two reasons: three additional cycles of latency for L2 accesses (compared to 1 cycle for ZC); and, the L1 and L2 capacities are reduced by 50% and 25%, respectively. Wilkerson did not report power overheads, thus we do our best to provide an estimate in Table 2. We ignore overhead due to ECC correction of repair patterns and the shifting layers along with their corresponding decoders. Wilkerson's method has a significant power overhead because of parallel access to both banks, and there is a high leakage power for the ST cells used for the tag array. Lastly, the area overhead of Wilkerson's method is modest, with ZC slightly higher. It should be noted that the area of L2 is around 41 times larger than L1. Consequently, *area* overhead of a protection scheme for the chip is mostly determined by the area overhead for the L2 cache. The 8T cell provides superior performance to either scheme, but at a cost of significant area overhead. The power overhead of the 8T L2 cache is also notably higher than the ZC design.

7.3 Significance

As we mentioned earlier, large on-chip caches are the major bottlenecks for enhancing process variation tolerance. Since the end of the free ride from clock scaling has already arrived, semiconductor companies need to use extremely conservative guard-bands for supply voltage and clock frequency to avoid

significant manufacturing yield loss. This has a major impact on the power consumption and operational frequency of modern microprocessors. In order to mitigate these effects, current microprocessors have already been equipped with coarse-grained redundancy schemes to protect the caches to the first order. An article by Hampson [15], reported that about 40% yield loss was observed when all forms of redundancy were removed from an Intel die. This was primarily due to the absence of redundancy from on-chip caches. In summary, ZC can be leveraged to allow full functionality while imposing overheads competitive with the best known alternatives.

8 Conclusion

Nanoscale CMOS technologies bring demanding reliability challenges to designers due to high degrees of process variation. In particular, SRAM structures are highly vulnerable to parametric alteration, thus the design of large on-chip caches that are both reliable and efficient is an important problem. In this work, we present ZerehCache, a flexible and dynamically reconfigurable cache architecture that efficiently protects on-chip caches in high failure rate situations. Our solution takes advantage of static multiplexing of the rows along with the added capability of dynamic word-line swapping to maximize the utilization of spare elements. Cache fault patterns are mapped to a graph coloring problem to configure the ZC architecture. We explored a large design space and came up with two suitable architecture configurations for L1/L2 ZCs such that they minimize the area and power overheads while achieving a desired level of robustness. An L1 ZC with 16% and an L2 ZC with 8% area overhead achieve yields of 99% and 96%, respectively. Finally, we compared our scheme with several conventional and state-of-the-art methods to illustrate its efficiency and effectiveness.

9 Acknowledgments

Our gratitude goes to the anonymous referees who provided excellent feedback on this work. This research was supported by ARM Ltd., the National Science Foundation grant CCF-0347411, and the Gigascale Systems Research Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.

References

[1] D. Achlioptas and C. Moore. The chromatic number of random regular graphs. In *8th International Workshop on Randomization and Computation*, pages 219–228, 2004.

[2] D. Achlioptas and A. Naor. The two possible values of the chromatic number of a random graph. In *Proc. of the 36th ACM Symposium on Theory of Computing*, pages 587–593, New York, NY, USA, 2004. ACM.

[3] A. Agarwal, B. Paul, S. Mukhopadhyay, and K. Roy. Process variation in embedded memories: failure analysis and variation aware architecture. *Journal of Solid State Circuits*, 49(9):1804–1814, 2005.

[4] 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 Very Large Scale Integration (VLSI) Systems*, 13(1):27–38, Jan. 2005.

[5] F. Aichelmann. Fault-tolerant design techniques for semiconductor memory applications. *IBM Journal of Research and Development*, 28(2):177–183, 1984.

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

[7] B. Berger and J. Rompel. A better performance guarantee for approximate graph coloring. *Algorithmica*, 5(3):459–466, 1990.

[8] S. Bhunia, S. Mukhopadhyay, and K. Roy. Process variations and process-tolerant design. In *Proc. of the 2007 International Conference on VLSI Design*, pages 699–704, Washington, DC, USA, 2007. IEEE Computer Society.

[9] B. Bollobas. The chromatic number of random graphs. *Combinatorica*, 8(1):49–55, 1988.

[10] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[11] L. Chang, D. Fried, J. Hergenrother, J. Sleight, R. Dennard, R. Montoye, L. Sekaric, S. McNab, A. Topol, C. Adams, K. Guarini, and W. Haensch. Stable sram cell design for the 32 nm node and beyond. *Symposium on VLSI Technology*, pages 128–129, June 2005.

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

[13] M. Franklin and K. K. Saluja. Built-in self-testing of random-access memories. *IEEE Transactions on Computers*, 23(10):45–56, 1990.

[14] R. Hamming. Error-detecting and error-correcting codes. *The Bell System Technical Journal*, 29(1):147–160, 1950.

[15] C. W. Hampson. Redundancy and high-volume manufacturing methods. *Intel Technology Journal*, 1(2), 1997.

[16] M. Horiguchi. Redundancy techniques for high-density drams. In *2nd Annual IEEE International Conference on Innovative Systems Silicon*, pages 22–29, 1997.

[17] L. D. Hung, M. Goshima, and S. Sakai. Seva: A soft-error- and variation-aware cache architecture. In *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*, pages 47–54, Washington, DC, USA, 2006. IEEE Computer Society.

[18] C. Kim, S. Sethumadhavan, M. Govindan, N. Ranganathan, D. Gulati, D. Burger, and S. W. Keckler. Composable lightweight processors. In *Proc. of the 40th Annual International Symposium on Microarchitecture*, pages 381–393, Dec. 2007.

[19] 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.

[20] W. Klotz. Graph coloring algorithms, 2002. Mathematik-Bericht 5, Clausthal University of Technology, Clausthal, Germany.

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

[22] I. Koren and Z. Koren. Incorporating yield enhancement into the floorplanning process. *IEEE Transactions on Computers*, 49:532–541, 2000.

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

[24] J. H. Lee, Y. J. Lee, and Y. B. Kim. SRAM Word-oriented Redundancy Methodology using Built In Self-Repair. In *IEEE International ASIC Conference '04*, pages 219–222, 2004.

[25] X. Liang, R. Canal, G.-Y. Wei, and D. Brooks. Replacing 6t srams with 3t1d drams in the l1 data cache to combat process variability. *IEEE Micro*, 28(1):60–68, 2008.

[26] T. Luczak. Chromatic number of random graphs. *Combinatorica*, 11(1):45–54, 1991.

[27] 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.

[28] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *IEEE Micro*, pages 3–14, 2007.

[29] D. Nassimi and S. Sahni. A self routing benes network. In *Proc. of the 7th Annual International Symposium on Computer Architecture*, pages 190–195, New York, NY, USA, 1980. ACM.

[30] S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou. Yield-aware cache architectures. *Proc. of the 39th Annual International Symposium on Microarchitecture*, 0:15–25, 2006.

[31] D. Roberts, N. S. Kim, and T. Mudge. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technol-

ogy. *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, pages 570–578, Aug. 2007.

[32] N. Sadler and D. Sorin. Choosing an error protection scheme for a microprocessor's l1 data cache. In *Proc. of the 2006 International Conference on Computer Design*. IEEE, 2006.

[33] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Varius: A model of process variation and resulting timing errors for microarchitects. In *IEEE Transactions on Semiconductor Manufacturing*, pages 3–13, Feb. 2008.

[34] K. Sasaki. A 9-ns 1-mbit cmos ram. *Journal of Solid State Circuits*, 24:1219–1225, 1989.

[35] Z. Shi and R. Lee. Implementation complexity of bit permutation instructions. In *Signals, Systems and Computers*, pages 879–886, Nov. 2003.

[36] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *Proc. of the 2004 International Conference on Dependable Systems and Networks*, pages 177–186, June 2004.

[37] K. Takahashi, H. Doi, N. Tamura, K. Mimuro, T. Hashizume, Y. Moriyama, and Y. Okuda. A 0.9 v operation 2-transistor flash memory for embedded logic lsis. *Symposium on VLSI Technology*, pages 21–22, 1999.

[38] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *Proc. of the 35th Annual International Symposium on Computer Architecture*, pages 363–374, June 2008.

[39] K. M. Thompson. Intel and the myths of test. *IEEE Journal of Design & Test of Computers*, 13(1):79–81, 1996.

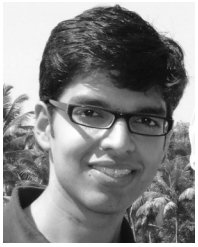
[40] N. Verma and A. Chandrakasan. A 256 kb 65 nm 8t subthreshold sram employing sense-amplifier redundancy. *IEEE Journal of Solid-State Circuits*, 43(1):141–149, Jan. 2008.

[41] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM*, 30(4):729–735, 1983.

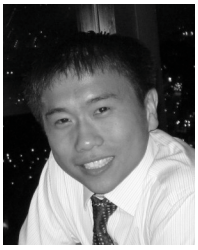
[42] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. Trading off cache capacity for reliability to enable low voltage operation. *Proc. of the 35th Annual International Symposium on Computer Architecture*, 0:203–214, 2008.

[43] X. Yang, M. Vachharajani, and R. B. Lee. Fast subword permutation instructions based on butterfly networks. In *SPIE, Media Processor 2000*, pages 80–86, 2000.

Amin Ansari is a Ph.D. candidate in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. His research interests include designing architectural and microarchitectural techniques for enhancing reliability of high-performance microprocessors in deep submicron technologies. Ansari received the B.S. degree in Computer Engineering from Sharif University of Technology, Iran in 2007 and the M.S.E. degree in Computer Science and Engineering from the University of Michigan in 2008. He is a student member of the IEEE and the ACM.



Shantanu Gupta is a Ph.D. candidate in the Electrical Engineering and Computer Science Department at the University of Michigan, Ann Arbor. His research interests span various aspects of compilers and architectures with a focus on fault tolerance, power-efficiency and single-thread performance. He received the B.Tech degree in Computer Science and Engineering from the Indian Institute of Technology, Guwahati in 2005, and M.S.E. degree in Computer Engineering from the University of Michigan in 2007.



Shuguang Feng is a Ph.D. candidate in the Electrical Engineering and Computer Science Department at the University of Michigan, Ann Arbor. His research interests include fault tolerant, reconfigurable computer architectures and investigating techniques that can exploit the interaction between software and hardware to enhance system reliability. He earned a Bachelor's Degree in Computer Engineering from the University of Florida in 2005, and received his M.S.E. degree in Computer Engineering from the University of Michigan, Ann Arbor in 2006.

Scott Mahlke is an Associate Professor in the Electrical Engineering and Computer Science Department at the University of Michigan where he leads the Compilers Creating Custom Processors research group. The CCCP group delivers technologies in the areas of compilers for multicore processors, application-specific processors for mobile computing, and reliable system design. Mahlke received the Ph.D. degree in Electrical Engineering from the University of Illinois at Urbana-Champaign in 1997. Mahlke's achievements were recognized by being named the Morris Wellman Assistant Professor in 2004 and being awarded the Most Influential Paper Award from the International Symposium on Computer Architecture in 2007.