# The Salvage Cache: A fault-tolerant cache architecture for next-generation memory technologies

Cheng-Kok Koh
School of Electrical and Computer Engineering, Purdue University
chengkok@ecn.purdue.edu

Weng-Fai Wong
Dept. of Computer Science, National Univ of Singapore
wongwf@comp.nus.edu.sg

Yiran Chen
Seagate Technologies
yiran.chen@seagate.com

Hai Li
Dept. of Electrical and Computer Engineering, Polytechnic Institute of NYU
hai.helen.li@ieee.org

***Abstract*— There has been much work on the next generation of memory technologies such as MRAM, RRAM and PRAM. Most of these are non-volatile in nature, and compared to SRAM, they are often denser, just as fast, and have much lower energy consumption. Using 3-D stacking technology, it has been proposed that they can be used instead of SRAM in large level 2 caches prevalent in today's microprocessors. However, one of the key challenges in the use of these technologies, such as MRAM, is their higher fault probabilities arising from the larger process variation, defects in its fabrication, and the fact that the cache is much larger. This seriously affect yield. In this paper, we propose a fault resilient set associative cache architecture which we called the *salvage cache.* In the salvage cache, a faulty cache block is sacrificed and used to repair faults found in other blocks. We will describe in detail the architecture of the salvage cache as well as provide results of yield simulations that show that a much higher yield can be achieved viz-a-viz other fault tolerant techniques. We will also show the performance savings that arise from the use of a large next-generation L2 cache.**

## I. Introduction

There is much ongoing research for alternative memory technologies. Many of the proposals, such as magnetoresistive RAM (MRAM), phase-change RAM (PRAM), and resistive RAM (RRAM) are non-volatile memory. Their high access speed, high density and low energy consumption make them replacement candidates for SRAM, especially in view of advances made in 3-D stacking technology [6].[1]

One of the issues that continue to be a challenge for using these next-generation memory technologies is their yield. Due to the material and manufacturing process involved, there is greater amount of defects and process variations. Coupled with sheer size and miniaturization involved, this can lead to poorer yields. The probability of a fault in a MRAM cell, for example, can be 2 to 10× that of an equivalent SRAM cell [13]. On the other hand, MRAM has much higher density. Using the same technology node, in the same area occupied by an SRAM cache, the capacity of the cache can be quadrupled if we use MRAM cells [6], [21]. This higher density allows for the use of architectural means to recover yield losses. In this paper, we introduce the *salvage cache*, a micro-architectural technique for tolerating faults in large, set associative caches. In a nutshell, a small number of faulty blocks in a cache set is sacrificed to repair other faulty blocks in the same set.

[1]Some of our references refer specifically to a variant of the MRAM called the spin torque transfer RAM (STT-RAM). We make no such distinction and shall refer to both as 'MRAM'.

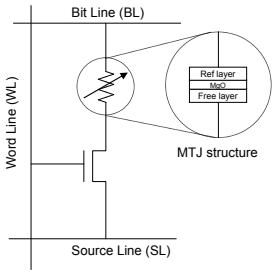


Fig. 1. 1T1J structure of STT-RAM cell.

Earlier works on fault-tolerant cache design generally took advantage of the inherent ability of especially set-associative caches to tolerate faults. In [14], [2], a cache block is turned off when they are found to be faulty. Other cache blocks in different ways of the same set of the disabled cache block are still functional. We call techniques such as these together as *down-sizing by discarding faulty blocks* (DDFB). There have been a number of variations of this theme including one for direct-mapped caches [1], the PADded cache [20], the 'Yield Aware Power Down' cache [17], and Intel Itanium's 'Pellston' technology [18]. Recently, two schemes called Word-disable (WDIS) and Bit-fix (BFIX) have been proposed [23]. 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%. In contrast, because of the higher fault probabilities, we are likely to find more than one faulty blocks in a set. This is especially so in modern L2 and L3 caches that have a large number of ways. The salvage cache exploits this fact and sacrifices an already faulty block to repair other faulty blocks in the same set.

## II. Magnetoresistive RAM

One of the more promising next generation memory technologies is the Magnetoresistive RAM (MRAM). The key component of MRAM is the *magnetic tunnel junction*

978-1-4244-5028-2/09/$25.00 ©2009 IEEE

(MTJ) [9], as shown in Fig. 1. A single MTJ is composed of a reference layer, a free layer and an oxide barrier layer (e.g. MgO). The magnetic direction of reference layer is fixed while the one of free layer can be changed by either a magnetic field or a polarized current. When the magnetization directions of reference layer and free layer is parallel (anti-parallel), the MTJ resistance is in low (high) resistance state. The high and low resistance states of MTJ, $R_{\max}$ and $R_{\min}$, are used to store information.

In *spin transfer torque RAM* (STT-RAM) [9], the magnetic direction of the free layer is changed by directly passing spin-polarized currents through MTJ, using the "1T1J" structure, where one NMOS transistor is connected to one MTJ, as shown in Fig. 1. The NMOS transistor is controlled by the word line (WL) signal. When reading data from a STT-RAM cell, a read current is injected to the selected memory cell, generating a voltage level $V_b$ at the bit line. At the same time, the same amount of current is injected to a dummy memory cell to generate a reference voltage level between the levels that can be generated by the memory cell with a MTJ of $R_{\min}$ or $R_{\max}$. One common practice for generating the reference voltage is to let the read current go through two connected memory cell with a MTJ of $R_{\min}$ and $R_{\max}$, respectively. Then the generated voltage level is divided by two as the reference voltage level ($V_{\text{ref}}$). We note that the resistance of MTJ is mainly determined by the thickness of MgO layer and the shape of MTJ, which are significantly affected by manufacturing variations. As a result, the $V_b$ generated by a MTJ $R_{\min}$ ($R_{\max}$) may be even higher (lower) than the $V_{\text{ref}}$. A read error may then occur. Such errors are permanent and necessitate correction mechanisms such as the salvage cache.

## III. Related works

There is a large body of literature dealing with process variations and yield especially at the circuit level [4], [3], [22], [5]. The yield of SRAM storage cells has also been the subject of several studies [15], [11]. We found three patents [14], [2], [7] that proposed circuits to bypass faulty cache blocks in a set-associative cache. More recently, Lee *et al.* [12] described a faulty cache simulation tool, CAFÉ as well as a variety of DDFB strategies to deal with faults. They also made a good case for the need to 'gracefully degrade' the cache in the presence of faults.

The Word-disable (WDIS) scheme and Bit-fix (BFIX) scheme proposed in a recent study [23] come close to being comparable with our proposal. In the WDIS scheme, two consecutive cache blocks are combined into a single cache block. A 64-byte cache block, for example, is divided into 16 32-bit words. With one bit to record whether a 32-bit word is functional, a 16-bit fault map is stored alongside the tag. In a read operation, two consecutive cache blocks are accessed simultaneously. Then, a four-stage shifter removes the defective word from a cache block based on the fault map to reconstruct one half of a full cache block. Consequently, the capacity of the cache is reduced by 50%.

Whereas the WDIS scheme performs the necessary repairs at the (32-bit) word level and within a cache block, the BFIX scheme performs the necessary patching for pairs of bits and stores the patches of 3 cache blocks in a cache block residing in another bank. A cache block that stores the patches is called a *fix line*, and it stores 10 patches for a cache block that it is paired with. Each patch contains the address of the faulty 2-bit group and the correct repair patterns. To protect against defects in a patch, a single-bit error correction code is used. In a read operation, the cache block that corresponds to a tag match and its fix line are accessed simultaneously. The patches are decoded and a 10-stage shifter removes defective 2-bit groups out of the tag-matched cache block. As a cache block can be used as a fix line for three other cache blocks, the BFIX scheme reduces the cache capacity by 25%. As we shall see, not only does the salvage cache produce better yield than WDIS and BFIX, it also results in caches with significantly higher average associativities.

Earlier, we had proposed the *buddy cache* [10] that pairs up two non-functional blocks to yield one functional block. A similar idea was proposed independently in [19]. The salvage cache significantly improves on this by using a single non-functional block to repair several others. Therefore, we did not compare the salvage cache against these proposals.

## IV. The Salvage Cache

Consider an $n$-way set-associative cache with $m$ sets. Let the $n$ blocks within a set be labeled $B_0$, $B_1$, ..., $B_{n-1}$. We divide each cache block, say $B_p$, into $k$ divisions, with each division guarded by a *fault bit* in a *fault map*. In other words, if a division is faulty, the corresponding fault bit for that division is set to 1. The fault bit of an operational division is set to 0. The tag field of a block is guarded by a separate bit. Therefore, there are $k+1$ fault bits for a block. In an $n$-way set-associative cache with $m$ sets, there are a total of $(k+1)nm$ bits in a fault map.

Let $f_{p0}f_{p1}...f_{p(k-1)}f_{pk}$, $f_{q0}f_{q1}...f_{qk}$, and $f_{r0}f_{r1}...f_{rk}$ denote the $k+1$ fault bits of cache blocks $B_p$, $B_q$, and $B_r$ and their tags, respectively. Suppose some of the $k$ divisions in the three blocks are faulty. Suppose that $f_{pi_1} = f_{qi_2} = f_{ri_3} = 1$. In other words, divisions $i_1$ of $B_p$, $i_2$ of $B_q$, and $i_3$ of $B_r$ are faulty. If $i_1 \neq i_2$, $i_1 \neq i_3$, and $i_2 \neq i_3$, there is a combination of the divisions of the blocks that will result in functional blocks. In particular, if we select $B_r$ to be the *victim* block, then we can use division $i_1$ and $i_2$ of $B_r$ in place of the respective faulty divisions in $B_p$ and $B_q$. $B_r$ will be disabled by setting $f_{rk}$ to 1.

A *victim map* is required to store the index of the victim block from whom replacement divisions are obtained. Given that there are $n$ blocks within a set, $\log n$ bits are required to encode the index of a block. Therefore, $n \log n$ bits are required to store the pairing information within a set. For simplicity, we assume that each set of the cache has a corresponding row in the fault map and victim map. We shall also assume that these two maps are placed adjacent to one another in a combined fault and victim map. We shall further assume that these maps are perfect. Assuming perfect maps also simplifies our exposition. Faulty maps can be dealt with in a way similar to the buddy cache. [10].

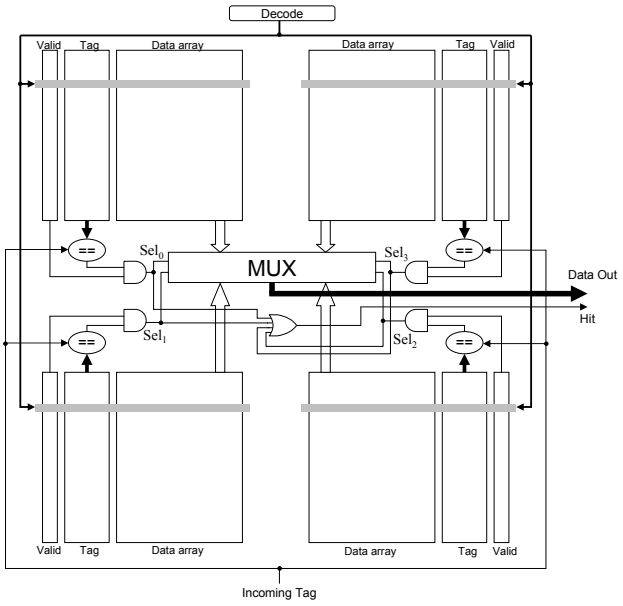


Fig. 2. A conventional 4-way set-associative cache.

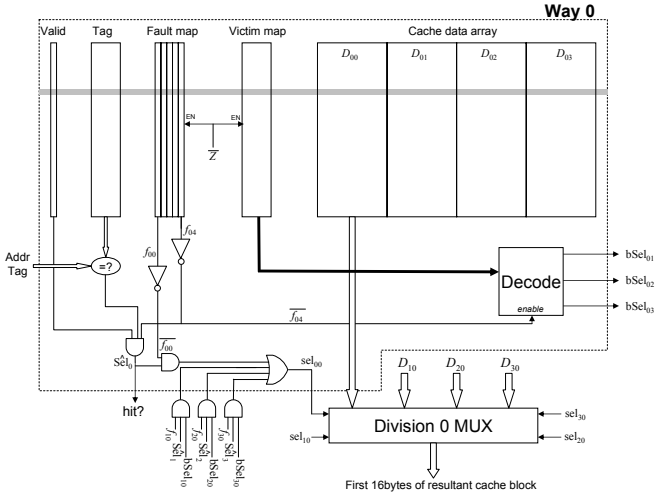


Fig. 3. Way 0 of a 4-division, 4-way set-associative, 64 bytes per block salvage cache.

For a fully operational cache block, its victim map entry simply points to itself. For a faulty block, if a victim block can be found within the set, the index of the victim block is stored in the victim map. If a victim block cannot be found, we assign ones to all the fault bits for that block in the fault map. In addition, the victim map entry is again pointed to itself. The initialization of the fault and victim map is done during the processor initialization using traditional built-in self test (BIST). This is a common assumption among fault tolerant cache architectures [1]. Details of how the victim map is configured is found in Section IV-A.

Figure 2 shows a conventional 4-way cache design. The signals from the decoder select the corresponding tag value and data block from four different ways in the same set. The result of comparing the stored tags with the incoming address tag together with the validity bit generate the selection signal $\text{Sel}_i$, $i \in \{0,1,2,3\}$. All four $\text{Sel}_i$ signals are sent into a 4-to-1 MUX to select the final output from the four data blocks. Also, if any $\text{Sel}_i$ signal is high, a 'hit' signal is asserted.

To implement a $k$ division salvage cache, we will make the following changes to an $n$-way, $l$ bytes per block set-associative cache (see Figure 3 – parts of the diagram has been omitted for clarity). In place of the $n$-to-1 $l$-byte MUX, $k$ $n$-to-1 $(l/k)$-byte MUXs are needed. Therefore, for every $(l/k)$-byte division, we have $n$ selection inputs $\text{sel}_{ij}$ to select $D_{ij}$, the $j$th division of block (way) $B_i$, $i \in \{0,\ldots,n-1\}, j \in \{0,\ldots,k-1\}$. Let $\text{Sel}_i$, $i \in \{0,\ldots,n-1\}$, be asserted when the incoming tag matches a valid stored tag $T_i$. To ensure that the matched tag is from an operational tag, we define $\hat{\text{Sel}}_i$ as $(\text{Sel}_i \cdot \overline{f_{ik}})$, where $f_{ik}$ is the fault bit of the tag. If any $\hat{\text{Sel}}_i$ signal in a set is high, a 'hit' signal is asserted.

Without loss of generality, we shall use the example of Way 0 in Figure 3 to explain the working of the salvage cache. Suppose selected cache set at Way 0 is not a victim. In addition, it has a functional division 0. Then $\overline{f_{00}} = 1$. When there is a hit on this set and way, $\hat{\text{Sel}}_0 = 1$. This in turn makes $\text{sel}_{00} = 1$, thereby making the Division 0 MUX select $D_{00}$.

Suppose that the selected cache set's Way 0 division 0 is faulty, i.e., $f_{00} = 1$. This, in addition to the fact that $f_{10} = f_{20} = f_{30} = 0$ by the way victims are selected, will ensure that $\text{sel}_{00}$ is not asserted. Suppose further that Way 1 contains the victim block. Then the victim map entry will decode to $\text{bSel}_{01}$. Together with $\hat{\text{Sel}}_0 = 1$, $\text{sel}_{10}$ will be asserted. In other words, $D_{10}$ will be selected to take the place of $D_{00}$.

If the selected cache set's Way 0 is the victim, then $\overline{f_{04}} = 1$, and $\hat{\text{Sel}}_0$ can never be asserted. In this case, $D_{00}$ will be used if some other way needs it. Note that if $D_{00}$ is faulty, the assignment makes sure that all other blocks that use this victim block has functional division 0's, and so $\text{sel}_{00}$ will never be asserted.

When the cache is faultless, we do not want to incur the overheads (especially the energy overhead) of this mechanism. We achieve that by using a single bit $Z$ as an enabling signal for the combined fault and victim map.

In our implementation of the salvage cache, we assume that like a conventional cache, it is protected from *transient* faults by ECC. In other words, the combined fault and victim map handles faults in next-generation memory cells due to defects in materials, lithography-based failures, etc. ECC deals only with soft errors.

Based on the PTM 45nm technology, the additional delay introduced by the remapping logic is 60ps, which is around 20% of a clock period for 3GHz frequency. In another word, the latency is increased by 1 cycle in the worst case, when compared to the baseline. The additional dynamic access energy is 0.024%, which is partly due to the 20,000 transistors in the remapping logic for a 8MB cache. In contrast, WDIS and BFIX incurred an additional 1 and 3 cycle latency, respectively. We do not have the energy

| Block | Divisions | Tag | Fault Map |
|---|---|---|---|
| 0 | $D_{00}D_{01}D_{02}D_{03}$ | $T_0$ | 0 0 1 0 1 |
| 1 | $D_{10}D_{11}D_{12}D_{13}$ | $T_1$ | 1 0 0 0 0 |
| 2 | $D_{20}D_{21}D_{22}D_{23}$ | $T_2$ | 0 0 0 1 0 |
| 3 | $D_{30}D_{31}D_{32}D_{33}$ | $T_3$ | 0 1 0 0 0 |

TABLE I
A CACHE SET IN A 4-WAY SALVAGE CACHE.

overhead of WDIS and BFIX, but the BFIX scheme requires around 26,000 transistors for a 2MB cache.

### A. Configuring the combined fault and victim map

Now, the remaining question is how the combined fault and victim map can be configured properly. As mentioned, the testing of memory cells in the cache (tag and data) is carried out during processor initialization using the traditional built-in self test (BIST) approach for memory as in [1]. First, the fault map stores the results of the BIST of each set of cache blocks (and tags). We will configure the victim map one cache set at a time.

The configuration of the victim map of a cache set can actually be formulated as a graph problem. Here, we form a compatibility graph for each cache set, where each vertex in the graph corresponds to a *faulty* block in the cache set. If the union of the functional data divisions of two faulty blocks results in a functional cache block (i.e., $k$ functional data divisions and a functional tag), we connect the corresponding two vertices in the graph, indicating that they are compatible and that they could be paired to form a functional cache block. If there exists a clique of $s \leq k+1$ vertices in the graph, these $s$ vertices can actually be partnered together to form $s-1$ functional blocks.

Consider a set in a 4-way cache, which has four faulty blocks as shown in the column labeled "Fault Map" in Table I. It is obvious that the union of any two of the faulty blocks results in a functional block (i.e., the bit-wise AND of the fault bits in fault map results in a sequence of all zero bits). These 4 blocks will form a clique of 4 vertices in the compatibility graph. It should be clear that we could use the functional data divisions $D_{00}$, $D_{01}$, and $D_{03}$ to replace the faulty data divisions $D_{10}$, $D_{31}$, and $D_{23}$, respectively, to produce 3 functional blocks. In this example, because block $B_0$ has a faulty tag, it is natural to victimize it. When every faulty block in a clique has a functional tag, it would be necessary to sacrifice one of them, say $B_i$, and turn it into a victim by disabling its tag, i.e., we set the fault bit $f_{ik}$ in the fault map to 1.

Before we formally state the problem of optimally identifying appropriate victims in a cache set, we first define the following: Let $c$ denote a clique, we define $v(c)$ to be the set of vertices in $c$. Note that the number of functional blocks that we can obtained from $c$ is $|c|-1$. Now, consider the corresponding compatibility graph $G=(V,E)$ of a cache set. If we partition the graph into a set of disjoint cliques, denoted as $C$, such that $V = \cup_{c \in C} v(c)$, the number of functional blocks we can obtained from this partitioning is $|V|-|C|$. Here, we assume that the disjoint partitioning may have one-vertex cliques. Therefore, the problem of optimally identifying appropriate victims in a cache set is equivalent to that of minimizing $|C|$. This is known as the minimum clique partitioning problem [8], which is difficult to solve.

Here, we propose a heuristic that could be implemented in hardware easily. First, we start with all faulty blocks that have a faulty tag. For each such victim block $B_i$, we find the lowest-indexed compatible imperfect cache block, i.e., the lowest-indexed beneficiary, as follows: We use bit-wise AND to determine whether $B_i$ and another faulty cache block $B_j$, $j \neq i$, can be combined to make $B_j$ functional. The $k+1$-bit output of the bit-wise AND go through a NOR gate, whose output is asserted if $B_i$ and $B_j$ are compatible. The output of the NOR gate is supplied to a priority encoder, which also takes in the output signals of other NOR gates performing compatibility check of other imperfect cache blocks with $B_i$. The priority encoder will select the lowest-indexed imperfect cache block that is compatible to $B_i$. Now that the lowest-indexed beneficiary has consumed some of the functional data divisions of victim block $i$, we set the corresponding fault bits of $B_i$ in the fault map to 1, and the beneficiary will no longer participate in the pairing process. We iteratively identify the next lowest-indexed compatible imperfect cache block for victim $B_i$ until we have set all fault bits of $B_i$ to 1 or a imperfect cache block compatible to $B_i$ cannot be found. At this point, we set all fault bits of $B_i$ to 1.

After we have considered all faulty blocks that have a faulty tag, we are left with faulty blocks that have functional tags. Start with the lowest-indexed imperfect cache block $B_i$, we disable its tag, i.e., we set the fault bit $f_{ik}$ to 1. We now repeat the procedure outlined in the preceding paragraph to identify all beneficiaries of $B_i$. We iteratively consider all other faulty blocks in the cache set until we have exhausted all faulty blocks that have functional tags.

## V. YIELD STUDY

In this section, we will study a limiting case of performance degradation, namely the catastrophic failure of the chip due to the failure of the cache. This is closely related to the 'yield' of the chip. Without loss of generality, we will use the following definition of yield: A set in a cache is functional if it has at least one functional way. (Even though this may appear to be too optimistic, we shall show in the next section the average associativity of functional caches, in particular, that of salvage cache, is much higher than 1.) A cache with $m$ sets is functional if after using part or all of its redundant resources, it has $m$ functional sets. The *yield* of a cache is the probability that it is functional. For this study, we targeted MRAM. However, the methodology presented here can also be applied to other memory technologies.

### A. Yield of MRAM

To investigate the probability of a MRAM read error, we considered the variation of the MTJ resistance. The variation

of MTJ resistance is assumed to be of a normal distribution. The TMR (Tunneling Magneto Resistance Ratio), i.e., $(R_{\max} - R_{\min})/R_{\min}$ is set to 100%. The difference between the means of $R_{\max}$ and $R_{\min}$ is set to 14.1× the deviation of $R_{\min}$ at 45nm [16]. The read error rate due to the MTJ resistance is computed to be about 0.11%. By taking into account the manufacturing defects, which is assumed to be the same level as the resistance-variation-induced read error rate, the total read error rate of STT-RAM is about 0.25%. This rate may increase as technology scales. Consequently, we considered fault probabilities of between 0 to 0.003.

### B. Yield comparison

Here, we present yield results obtained both analytical yield model as well as Monte Carlo simulations for the following four types of caches: Salvage cache, DDFB cache, WDIS cache, BFIX cache. The purpose is to cross-validate the yield results obtained from both methods. The Monte Carlo simulations were performed for fault probabilities $p_b$ ranging from 0.000 to 0.003, with increments of 0.0005. For each fault probability $p_b$, we assume that occurrences of faults in the caches are uniformly random. We randomly inject faults (with probability $p_b$) into four hundred instances of 8 Mbyte 32-way set-associative caches and then evaluate whether a cache is functional based on the four fault-tolerant schemes. In the case of salvage cache and DDFB cache, we also inject faults into both tag array and data array. For WDIS and BFIX caches, we only injected faults into the data arrays.

Due to space constraint, however, we shall present only the details of the analytical yield model for the salvage cache; analytical yield models for other types of caches can be derived in a similar fashion. We shall use the following notation: An $n$-way set-associative cache with $m$ sets, with tag size $t$ and $k$ data divisions, each of which has $d$ data bits.

The probability that a cache block (both tag and data) is faulty is $P_{\text{block-faulty}} = (1-(1-p_b)^{t+dk})$. For two cache blocks to be paired in the salvage cache, at least one of the two corresponding tags must be operational. Moreover, the data divisions at the same position in the two cache blocks cannot be both faulty. Therefore, the probability that the pairing of two cache data blocks in a set produces an operational block is $P_{\text{paired-block}} = (1-(1-(1-p_b)^t)^2)(1-(1-(1-p_b)^d)^2)^k$. Within a set, there are altogether $\binom{n}{2}$ possible pairs of cache blocks. As defined, a set is non-functional only if all blocks are faulty and none of these pairings can produce an operational cache block. Hence, the probability that a set is functional is $P_{\text{set}} = 1 - P_{\text{block-faulty}}^n (1 - P_{\text{paired-block}})^{\frac{n(n-1)}{2}}$. For a cache with $R$ redundant sets, at most $R$ sets can be faulty for the cache to be operational. The probability that the salvage cache is operational is therefore

$$P_{\text{SC}} = \sum_{i=0}^{R} \binom{m+R}{i} P_{\text{set}}^{m+R-i} (1-P_{\text{set}})^i.$$

For a 32-way 8 Mbyte salvage cache with a block size of 64 bytes divided into 8 data divisions, $P_{\text{SC}}$ is effectively 1.00 even when the fault probability is as high as 0.003 and $R = 0$ as shown in Figure 4. In other words, the salvage cache is extremely robust because of the abundance of pairing options available. It is also evident from Figure 4 that the analytical model tracks the results from Monte Carlo simulations of 8 Mbyte salvage cache with 8 data divisions very well, thereby cross-validating each other.

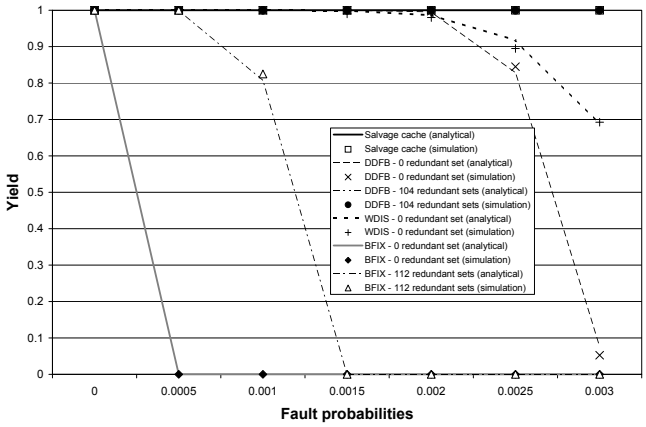


Fig. 4. Yield results of various caches obtained from analytical yield models and Monte Carlo simulations.

As we can see from Figure 4, with 0 redundant cache set, the yield of DDFB cache is not competitive. However, the yield can be improved significantly when 104 redundant cache sets are added. Note that we use 104 redundant sets in the DDFB cache so that the total number of memory cells in the fault map, tag array, and data array of a DDFB cache is similar to that in the fault map, victim map, tag array, and data array of a salvage cache. In other words, they have the same memory cell count.

In Figure 4, we also show the yield of 8 Mbyte 32-way set-associative caches implemented with the WDIS and BFIX schemes. The WDIS scheme has almost perfect yield even with 0 redundant cache set. Note that we do not have to add redundant cache sets to WDIS cache because the fault map of WDIS has more memory cells than the fault map and victim map of a salvage cache combined and than the fault map of a DDFB cache. With 0 redundant cache set, the yield of the BFIX scheme drops quickly to 0 as the fault probability increases. As shown in Figure 4, the yield of the BFIX scheme can be improved with 112 redundant cache sets, which are added to maintain the same memory cell count as a salvage cache. However, the yield still drops to 0 as the fault probability increases beyond 0.001. Therefore, even though the BFIX scheme in theory may have an effective associativity of 24 for a 32-way set-associative cache, we may not have a functional BFIX cache in practice. We believe that the yield of the BFIX scheme can be improved with triple modular replication of the patches. However, that is beyond the scope of this work. It is important to keep in mind in both WDIS and BFIX caches, we assume that the tag array is perfect, as did [23].

In summary, the salvage cache has better yield than the DDFB cache and the BFIX cache, and similar yield compared to the WDIS cache, even though the WDIS cache

| | 4MB L2 | | 8MB L2 | |
|---|---|---|---|---|
| | Conv. | Salv. | Conv. | Salv. |
| Read lat. (ns) | 2.59 | 2.65 | 3.28 | 3.34 |
| Remap. lat. (ns) | N/A | 0.06 | N/A | 0.06 |
| Write lat. (ns) | 2.59 | 10.83 | 3.28 | 11.10 |
| Read en.(nJ) | 0.49 | 0.50 | 0.73 | 0.73 |
| Remap. en. (nJ) | N/A | 0.0064 | N/A | 0.0064 |
| Write en. (nJ) | 1.72 | 1.73 | 1.98 | 1.99 |
| Leak. pow. (mW) | 104.23 | 104.24 | 126.66 | 126.68 |

TABLE II

LATENCIES AND ENERGIES OF OUR CONVENTIONAL AND SALVAGE CACHE MODELS.

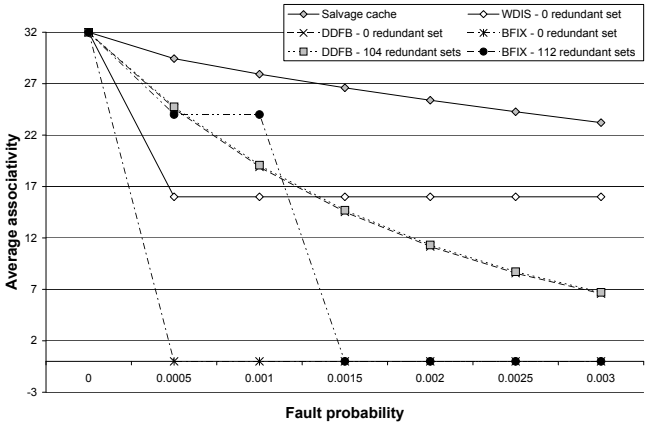


Fig. 5. Average associativity of salvage cache vs. DDFB cache, WDIS cache, and BFIX cache.

has an advantage of having a perfect tag array. In the next section, we will demonstrate the advantage of the salvage cache in terms of average associativity, which translates into better throughput performance of the salvage cache over other types of caches.

## VI. ENERGY CONSUMPTION

In our work, we modified the CACTI based on the parameters of the MRAM based cache design in [21] and [6] by carefully scaling the technology node to 45nm. The energy and latency parameters of 4MB and 8MB, 32-way, 64-byte block size sequential access cache are shown in Table II. Here we assume the write pulse duration of MRAM cell is 10ns. The increase in write latency is handled easily by mechanisms such as write buffering in the processor. Otherwise, the overhead of the salvage cache, both in terms of latencies and energies, are minimal.

## VII. PERFORMANCE STUDY

The key advantage of salvaging is that more functional blocks are made available. Figure 5 shows the average associativity (i.e., the average number of functioning blocks per set) of the 8 Mbyte salvage cache for various fault probabilities, ranging from 0 to 0.003, obtained through Monte Carlo simulations. The average associativities of the

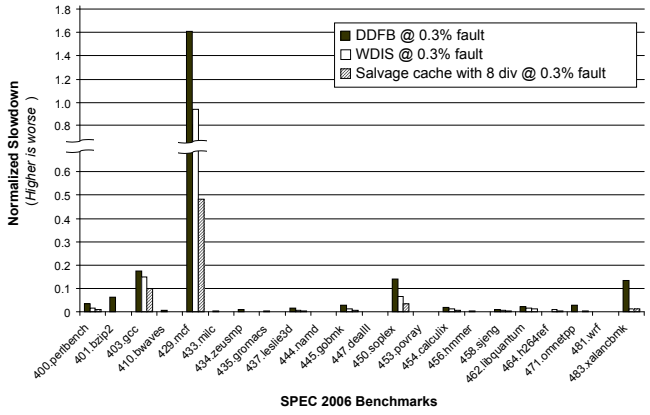


Fig. 6. Normalized slowdown for 22 SPEC2006 benchmarks.

DDFB cache with 0 and 104 redundant cache sets are also plotted. We can observe that increasing the number of redundant cache sets improves the average associativity only marginally, although it improves the yield substantially.

Recall that a cache implemented with WDIS and BFIX schemes have 50% and 75% of associativity, respectively, when the cache is functional. As the WDIS cache has 100% yield over the range of fault probabilities 0 to 0.003, the average associativity is 16 for all non-zero fault probabilities, as shown in Figure 5. For the BFIX cache, the average associativities are shown to be zero for fault probabilities that are too high for the BFIX cache to be functional.

As can be seen in Figure 5 the salvage cache configurations are better able to ameliorate the negative impact of increasing fault rates. At higher fault probabilities, the salvage cache significantly outperforms DDFB. At a bit fault probability of 0.003, DDFB (WDIS) can achieve only about 29% (69%, respectively) of the associativity attained by salvaging. While the salvage cache in theory has lower associativity than the BFIX scheme at high fault probabilities, the BFIX scheme does not produce a functional cache when the fault probability is higher than 0.001 (see Figure 4).

We evaluated the performance degradation under the DDFB and our salvage cache using the cycle accurate x86 simulator PTLsim [24]. The experiments were configured to simulate a 4-wide issue, dispatch, writeback, and commit out-of-order Intel Core processor with 128-entry reorder buffer, 256 physical registers, 2 load units, 4 integer ALU, 1 floating point adder, 1 floating multiplier and 1 floating point conversion unit. The fetch, load and store queues have 32, 32, and 48 entries, respectively. There are 5 pipeline stages for the front-end and 16 branches in flight can be accomodated.. All the configurations in our experiments had a 32KB, 8-way, 64-byte block size L1 data cache, and a 32KB, 8-way, 64-byte block L1 instruction cache. The load hit latency is 2 cycles. The L2 of our baseline configuration is a 8MB, 32-way, 64-byte block size sequential access cache with a 13 cycle hit latency. The other configurations used for comparison were a DDFB, WDIS and a salvage cache with 8 divisions, all with a fault probability of 0.003 in a 8MB L2

cache. BFIX has zero yield at this probability and so is not considered. In these, all cache parameters were the same as those of the baseline except that the hit latency is 14 cycles - one extra to account for the overhead of the various schemes.

We used 22 of the 29 SPEC2006 benchmarks. For some unknown reasons, the remaining benchmarks failed to run correctly on the 'vanilla' version of PTLsim. In all 22 benchmarks, we instrumented the code such that PTLsim full simulation is started only after data initialization. Simulation is stopped after simulating 2 billion x86 instructions. Depending on the instruction mixes of the benchmarks this can mean anywhere from less than 2 billion (due to out-of-order integer execution) to nearly 30 billion cycles (for floating point intensive code).

Figure 6 shows the results of our processor simulation in terms of performance slowdown. Due to the very large design space and the high cost of simulating each design point (averaging 12 wall-clock hours per run), we can only simulate certain design points. We have chosen the worst fault probabilities, namely 0.03% fault, for DDFB and the salvage cache (with 8 divisions). From Figure 6 we see that for many of the benchmarks there is little performance impact: even the reduced L2 caches are sufficient to accommodate the working set of the benchmarks. For large caches, ideally one should perform full system SMP simulator. However, the cost of each run is too high for us to get try out even a reasonable section of the design space. Besides, there are yet to be widely accepted benchmarks for a SMP setting. Even so, we see that for benchmarks with large working sets such as `400.perlbench`, `401.bzip2`, `403.gcc`, `429.mcf`, `450.soplex`, and `483.xalancbmk`, we see significant performance degradation for DDFB. For `429.mcf`, DDFB showed a 160% slowdown relative to the baseline, WDIS showed a 95% slowdown while the salvage cache's slowdown was 48%. This is evidence that the salvage cache, besides having better yield, also performs better than DDFB and WDIS because of its higher effective associativities.

## VIII. Conclusions

In this paper we introduce the salvage cache. It is a cache architecture that tolerate faults by recycling faulty blocks to patch one another in a set. Although the idea is generally applicable to all types of caches, it is particularly attractive when the fault probabilities are high. Otherwise, the chance of finding more than one faulty block to form the victim-beneficiary relationship would be low. Ideally, we would want every functional division of the victim to be used to patch some other faulty block. High fault probabilities have been observed in MRAM, thus making the salvage cache a promising structure for implementing large MRAM caches.

We evaluated the salvage cache against other alternatives of introducing fault tolerance. Our results show that the salvage cache gives significantly higher yields as well as higher effective associativity. The latter results in better performance. We therefore conclude that the salvage cache structure is ideal for implementing a large set-associative L2 cache with next-generation memory technologies.

## 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 Sys.*, 13(1):27–38, Jan 2005.

[2] R. K. Arimilli, J. S. Dodson, J. D. Lewis, and T. M. Skergan. Cache array defect functional bypassing using repair mask. U.S. Patent Number 5,958,068, 1999.

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

[4] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *40th DAC*, pp. 338–342, 2003.

[5] S. H. Choi, B. C. Paul, and K. Roy. Novel sizing algorithm for yield improvement under process variation in nanometer technology. In *41st DAC*, pp. 454–459, 2004.

[6] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen. Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. pp. 554–559, 2008.

[7] Y. Fujimoto. Cache memory having flags for inhibiting rewrite of replacement algorithm area corresponding to fault cell and information processing system having such a cache memory. U.S. Patent Number 6,145,055, 2000.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability*. 1979

[9] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano. A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram. In *IEDM '05*. pp. 459–462. 2005.

[10] 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 Arch. and Code Opt.*, 6(2):1–34, Jun 2009.

[11] F. J. Kurdahi, A. M. Eltawil, Y.-H. Park, R. N. Kanj, and S. R. Nassif. System-level SRAM yield enhancement. In *7th ISQED*, pp. 179–184, 2006.

[12] H. Lee, S. Cho, and B. R. Childers. Performance of graceful degradation for cache faults. In *ISVLSI '07*, pp. 409–415, 2007.

[13] H. Li, and Y. Chen. An Overview of Nonvolatile Memory Technology and The Implication for Tools and Architectures. In *DATE '09*, pp. 731–736, 2009.

[14] D. McClure. Method and system for bypassing a faulty line of data or its associated tag of a set associative cache memory. U.S. Patent Number 5,666,482, 1997.

[15] S. Mukhopadhyay, H. Mahmoodi, and K. Roy. Statistical design and optimization of SRAM cell for yield enhancement. In *ICCAD '04*, pp. 10–13, 2004.

[16] K. Nam, S. Oh, W. Kim, J. Jeong, D. Kim, S. Lee, Y. Kim, K. Kim, J. Lee, I. Yeo, U. Chung, and J. Moon. Electrical and magnetic properties of defect-mediated magnetic tunnel junctions using MgO. In *MMM '08*, 2008.

[17] S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou. Yield-aware cache architectures. In *MICRO 39*, pp. 15–25, 2006.

[18] Intel's newest Quad Xeon MP versus HP's DL585 Quad Opteron. http://www.anandtech.com/cpuchipsets/intel/showdoc.aspx?i=2872&p=2, Nov 2006.

[19] D. Roberts, N. S. Kim, and T. Mudge. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology. In *10th DSD*, pp. 570–578. 2007.

[20] P. P. Shirvani and E. J. McCluskey. PADded cache: A new fault-tolerance technique for cache memories. In *17th VTS*, pp. 440–406. 1999.

[21] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen. A novel 3D stacked MRAM cache architecture for CMPs. In *HPCA '09*, pp. 239–249. 2009.

[22] J. Tschanz, K. Bowman, and V. De. Variation-tolerant circuits: circuit solutions and techniques. In *42nd DAC*, pp. 762–763, 2005.

[23] 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. *Comput. Archit. News*, 36(3):203–214, 2008.

[24] M. T. Yourst. PTLsim: A cycle accurate full system x86-64 microarchitectural simulator. In *ISPASS '07*, pp. 23–34, 2007.