Cost-Quality Trade-offs of Approximate Memory Repair Mechanisms for Image Data

Qianqian Fan, Sachin S. Sapatnekar and David J. Lilja
Department of Electrical and Computer Engineering, University of Minnesota
Minneapolis, MN 55455
e-mail: {fanxx297, sachin, lilja}@umn.edu

Abstract

The traditional approach for increasing yield in large memory arrays has been to eliminate all hard errors using repair mechanisms. However, the cost of these mechanisms can become prohibitive for cheaper memories, which have higher error rates. Instead of completely repairing faulty cells, this paper introduces new approximate memory repair mechanisms that only partially repair both CMOS DRAMs and STT-MRAMs. By combining redundant repair with unequal protection, such as skewing the limited spare elements available for repairing faults towards the *k* most significant bits, and a hybrid bit-shuffling and redundant repair scheme, the new mechanisms maintain excellent output quality while substantially reducing the cost of the repair mechanism, particularly for increasingly important cluster faults.

Keywords

Approximate memory repair, unequal protection, CMOS DRAMs, STT-MRAMs.

1. Introduction

Several decades of device scaling has substantially reduced the cost per bit of memories, yet yield loss still remains a significant issue. Furthermore, process technology is still evolving for emerging memories, such as spin-transfer-torque magnetic RAMs (STT-MRAMs), which further impacts their yield. Hard errors in CMOS dynamic RAMs (DRAMs) typically dominate over soft errors [1], [2], while the emerging nature of STT-MRAM technology suggests that process-related hard errors are more likely than transient retention errors. The traditional approach to handling memory faults, and thereby improving yield, has been to root them out completely using various off-line redundancy mechanisms, such as spare rows and columns [3]-[5]. These repair schemes define yield as the probability that *no faults* occur [6]. As the probability of failure increases, though, these conventional redundant repair approaches will lead to tremendous increases in overhead to repair the memory to 100 percent correctness.

For a set of emerging applications, a new opportunity has arisen for coping with errors instead of completely repairing them. This paper addresses image applications, which show inherent error-tolerance. We employ the notion of approximate memory [7], which exploits the idea that some memory errors are not only nonfatal, but can be leveraged to enhance power and performance with minimal loss in quality [8].

The economics of memory dictate that the cost of memory increases nonlinearly with reliability. As a result, memories with few errors are very expensive, and the cost reduces greatly with higher error rates. We develop an approach to employ these lower-cost, higher-error memories for image applications.

The application space of this work is related to the notion that large amounts of image data are created and stored today, but most of these images are seldom viewed or used. For a multimedia image appliance, such data could therefore be stored in lower-cost "cold storage" systems, similar to those used by major social media networks or image sites. Our work ensures that even these lower-cost, lower-reliability memories produce results where the quality degradation is controlled, leading to an opportunity to trade off cost with storage quality.

We evaluate a set of *approximate memory repair* mechanisms that only partially repair memory to reduce the cost of the repair mechanism at the expense of some loss in output quality. We extend prior conventional memory repair methods that use limited spare elements to several new schemes, including the *k*-MSB redundant repair scheme that skews the redundant repair elements to the first *k* most significant bits of a byte, and compressed bit-shuffling which reduces the overhead of prior bit-shuffling approaches [8] while maintaining comparable quality. We further show that the bit-shuffling related schemes do not perform well in the presence of row, column, and cluster faults, which are important in real world applications [4], [9]. We also propose a new hybrid bit-shuffling and redundant repair scheme.

Our results on a representative image compression application demonstrate that these approximate memory repair mechanisms make repair possible even in memories at the low end of the yield spectrum while maintaining good output quality, and are effective in tolerating cluster faults. The results also show which mechanisms should be used when given a specific cost (area) budget or specific quality constraints. A notable contribution of this work is our comprehensive models of the area and delay overheads of each approach. To our knowledge, this is the first work to present such complete models. We further point out that the principles are applicable to other applications that admit approximation.

2. Memory organization and fault models

2.1. Data array organization

Our memory model is based on the structure in CACTI [10], as depicted in Figure 1. The memory is organized into multiple identical banks, each of which can be accessed in parallel, chosen by the bank address input. A bank is composed of multiple identical subbanks, which are further divided into multiple mats. All mats are activated during an access, and bits within a word are interleaved across mats. A mat has four

subarrays that share the predecoding and decoding circuitry, and each subarray has its own associated peripheral circuitry.

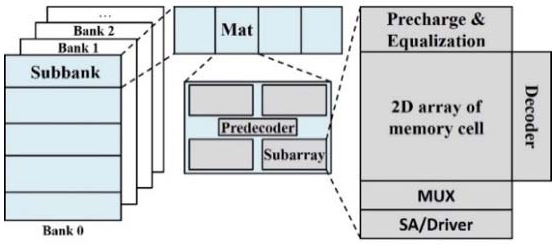

Figure 1: Organization of the memory array [10].

2.2. Fault models

2.2.1. Fault distribution

If λ is the mean number of faults in a memory array of size M, the failure probability of a bit-cell is $P_{cell} = \lambda/M$. The probability of n failing cells in the array follows the binomial distribution. This approximation holds for our typical use case where M is very large and P_{cell} is small [6]. A sample from this distribution provides the total number of hard faults in a specific memory array. In our experiments, we allocate this total number of faults to be *single cell failures* that affect an isolated cell, *column failures*, *row failures*, and *cluster failures*, which affect an entire column, row, and cluster, respectively [11][13], according to a set probability. We further set the fault type to a specific functional fault according to a user-specified probability, where the set of fault types for DRAMs and STT-MRAMs are described in the rest of this section.

2.2.2 DRAM fault types

The functional degradations that could result from a DRAM fault can be simplified into several categories [12]. For *stuck-at faults* (SAFs), the value in the cell is fixed at logic 1 or logic 0, while for *stuck-open faults* (SOFs), the cell is completely inaccessible. In *transition faults* (TFs), a rising or falling transition cannot be realized (these are similar to SAFs, but the difference is that the stuck-at logic level is unknown). *Coupling faults* may be manifested as *inversion faults* (CFin), where a transition in one cell will invert one of its neighbors, *idempotent faults* (CFid), where a transition in one cell will set one of its neighbors to a fixed value, or *state faults* (CFst), where a specific value in one cell will set one of its neighbors to a fixed value. More comprehensive fault models [13] can also be applied but they are unlikely to introduce any meaningful changes for final results.

2.2.3. STT-MRAM fault types

Failures in STT-MRAMs [5] can be classified into persistent errors, which are similar to hard faults, and transient errors, which are similar to soft errors. We focus here on persistent errors, primarily those caused by process variations. The behavior of these errors can be expressed in several ways. *Transition faults* (TF0/TF1) are write errors that are caused by insufficient MTJ write current, which prevents a switching event from being completed. *Read disturb faults* (RDFs) are caused when the read current is too high and inadvertently flips the cell value during a read operation. *Incorrect read faults* (IRFs) occur when, due to insufficient read sense margin, an incorrect value is read from the cell.

3. Memory repair schemes

3.1. Conventional repair schemes

3.1.1 Two-dimensional (2D) redundant repair

Two-dimensional redundant repair, illustrated in Figure 2, involves the insertion of spare rows and/or columns into a memory array. When a faulty cell is detected, a spare element may be used to replace it (spare elements have the same failure probability as the main memory array, and if the spare element has a fault, it cannot be used to repair faulty memory elements).

Given a fault map within a memory, which is typically obtained using a memory test, various algorithms are available to determine how a spare row/column can be used to replace the row or column in which the fault lies. In our experiments, we apply the widely used essential spare pivoting (ESP) redundancy analysis algorithm [14] for this purpose.

3.1.2. Segmented memory repair

When an entire row (column) is used to repair a faulty row (column) containing only one or a small number of faulty cells, a large number of spare rows and columns may be required to ensure full repair. Further flexibility may be provided by dividing the spare rows and columns into segments [4]. If a memory column has a single fault in a specific row, only the segment of a spare column corresponding to that row is used, and other segments may be deployed to repair other faults.

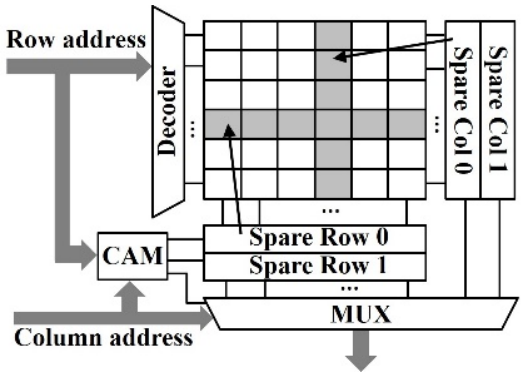

Figure 2: Two-dimensional redundant repaired memory.

3.2. Repair for approximate memories

The schemes in Section 3.1 were originally designed to replace faulty memory cells with spare functional cells with the goal of achieving full repair with high probability. However, for error-tolerant applications, full repair is not necessary and partial or approximate repair may be adequate. In this section, we propose a set of schemes that can be used to partially repair faults to provide an approximate memory for such applications.

For image compression applications, we will evaluate these schemes by using the power signal-to-noise ratio (PSNR) as a quality metric. The key idea of nonuniform protection is to protect some bits more carefully than others. For example, the role of higher-order bits is more significant than that of lower-order bits in determining a quality metric such as PSNR.

3.2.1. Limited spare rows and columns

One extreme end of this partial redundancy continuum corresponds to *unprotected memory* with no redundancy whatsoever. This approach has zero overhead and can be

expected to provide the lowest-quality result. At the other extreme is the notion of full repair, as discussed in Section 3.1. This approach will have the best quality, but at the highest cost. Intermediate points correspond to different cost-quality trade-offs. For approximate applications, we can limit the number of spare rows and columns of conventional repair schemes to reduce the area and delay overhead in the memory, while delivering adequate accuracy to maintain the quality metric.

3.2.2. *k*-MSB repair

The *k*-MSB repair scheme hybridizes the redundant repair scheme with the notion of nonuniform protection. This scheme is based on the observation that the cost of providing protection increases with the number of bits, but its effectiveness in improving the system quality metric goes down steeply after the first few bits. Therefore, this method applies more spare elements to protect the first *k* MSBs of the byte possibly even achieving full repair for these bits. A lower level of protection could still be applied to the LSBs by using some of the spare elements since they may still be vulnerable to clustered faults. The number of available spare elements is determined by the probability of the clustered faults. As for the implementation, different threshold values of the ESP algorithm can be set for different levels of protection. For a low level of protection, a larger threshold allows the faulty rows, faulty columns, and other types of cluster faults to be prioritized for repair using these limited spare elements. As we will show, this scheme can greatly reduce the overhead with minimal quality impact.

3.2.3. Bit-shuffling

The basic bit-shuffling scheme was proposed in [8]. Like our *k*-MSB method, it leverages the notion that higher-order bits are more critical to a computation than lower-order bits. This mechanism rotates the bits in a word in case of a fault to ensure that higher-order bits are protected. When data is read from or written into memory, it is rotated so that the higher-order bits are stored in a fault-free location, and the fault is effectively moved to a lower-order location.

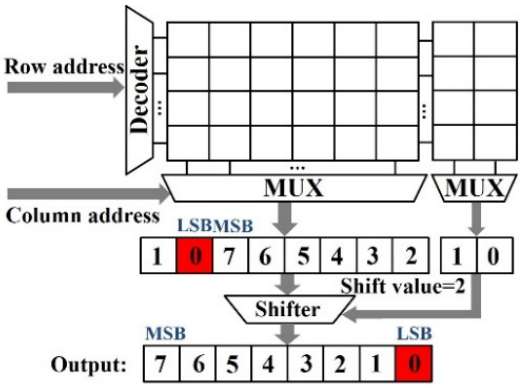

Figure 3: A schematic of the bit-shuffling scheme for the read operation.

For our image compression application, the length of each item of image data is 1 byte. Given a fault map, if the n^{th}-most significant bit has a fault, then the method uses a circular shifter to rotate the data to the left by *n* places as it is stored into memory. This operation implies that the LSB of this byte is placed into a faulty cell. A variation of the basic scheme is characterized by a parameter, n_{FM}, which decides the maximum number of shifts that are permitted. The largest shift is limited to $(2^{n_{FM}}-1)$, and therefore the $(2^{n_{FM}}-1)$ highest-order bits are protected. For a byte, a value of 1, 2, or 3 for n_{FM} corresponds to a shift of up to 1 bit, 3 bits, or 7 bits. A lower value of n_{FM} reduces the area and delay overheads of the scheme but limits its correction capability.

Figure 3 illustrates an implementation of this scheme. Here each row contains *b* bytes, and there are bn_{FM} columns to the right of the main storage array. For the image application, this configuration allows each byte to be rotated to preserve the correctness of its MSB. The MUX at right, below this auxiliary array, chooses the rotation requirements of the byte of interest within a line, and its select input is the column address. In this example, n_{FM} =2, which enables the three MSBs to be protected. If the second MSB has a fault, then the shift value is two ($(10)_2$). While storing data, a right-circular shift translates the data by two bits to place the LSB in the faulty cell, which is shown as the stored value. The figure illustrates a read operation, where the opposite operation-a left circular shift-is performed to restore the MSB to its rightful location.

3.2.4. Compressed bit-shuffling

The bit-shuffling scheme in [8] was implemented using an additional set of n_{FM} bits for each line. For the image compression application, the relevant RGB data is only a byte long, implying the overhead of (bn_{FM}) per line is prohibitive. The discussion in [8] did not encounter this problem since the word size was substantially larger, which allowed the overhead to be amortized over a larger number of data bits.

To overcome this large overhead, we propose a modification to this bit-shuffling scheme. In a realistic scenario where the number of faults is not large, most of the rows and columns will require no correction, implying that the rotation requirements stored in n_{FM} will be zero in most cases. Therefore, we propose a storage scheme with a compression rate of 2^q where a single shift value is shared by a set of 2^q bytes. As a result, the overhead of storing the rotation bits is reduced from bn_{FM} per line to $(b/2^q)n_{FM}$. The drawback of this case is that, if there are two faults within a block of 2^q words that share a shift value, only one can be corrected.

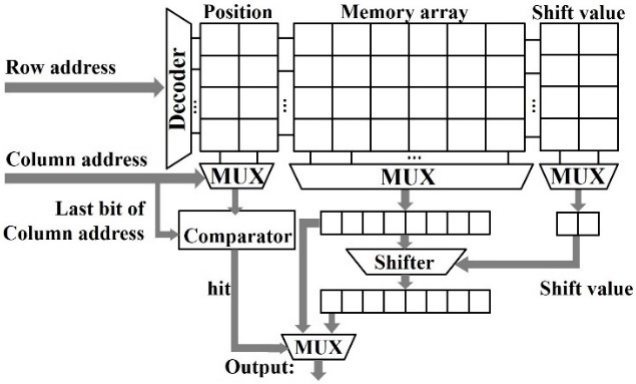

Figure 4: An improved implementation of the bit-shuffling scheme.

The hardware implementation of this scheme is illustrated in Figure 4. An additional array is placed to the left of the

memory array, storing the position within the 2^q-bit block of the word that has a fault, if any. As before, its shift value is stored to the right of the memory array. The column address, minus the bottom q bits, is applied to the MUXes connected to the *Position* and *Shift Value* arrays. The lower order bits represent the location of the byte within the line. If they match the position for that line, the corresponding shift value is used to rotate the stored data, if necessary.

3.2.5. Hybrid bit-shuffling and redundant repair

A major drawback of bit-shuffling-based schemes is that they work well only for isolated faults. In reality, we may see row, column, or clustered faults [4], [9] where faults strike multiple adjacent cells. The probability of having more than one fault per word is then greatly increased, even when words are interleaved, and bit-shuffling can be ineffective.

Redundant repair schemes with spare rows or columns are well suited to handle such errors, however. If there is a row/column error, the entire row/column is replaced. Therefore, we combine the redundancy and the bit shuffling schemes to maximize the repair efficiency. A low level of redundant repair protection is applied by using a limited number of spare elements to repair row, column, or cluster faults. Bit-shuffling is additionally used to effectively deal with isolated faults. As for the implementation, the address is sent to a CAM to perform the redundant repair operation while, in parallel, the shift value is used for the bit-shuffling repair, as shown in Figure 5. Since this scheme is a combination of these two schemes, the overhead is the sum of the individual overheads of the two constituent schemes.

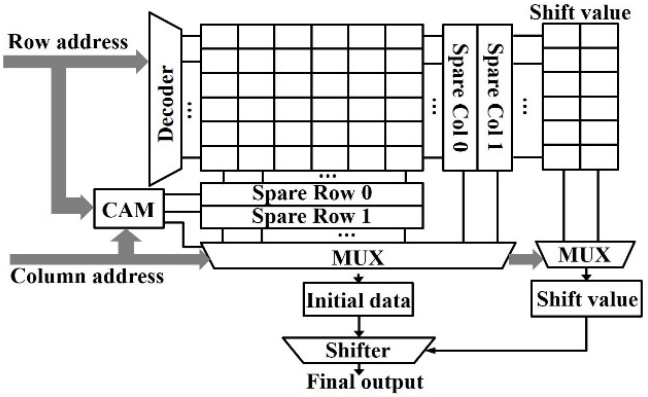

Figure 5: Implementation of the hybrid scheme.

4. Area and Delay Overheads

4.1. Area model

The area model for each scheme is based on the transistor count of the structure. For both the 1T1C DRAM and the 1T1MTJ STT-MRAM, the area is proportional to the transistor count (the MTJ cell area is dominated by the access transistor). Therefore, similar area models are used for both memory types.

4.1.1. Basic structure

As described in Section 2, the memory is eventually divided into several identical mats, each of which has a grid of 2 × 2 subarrays. Consider a mat with *M* rows, *N* columns with an output width of *W*, as shown in Figure 6. Each mat consists of:

- A row decoder and wordline drivers with *M* outputs, with an area cost of *6M*, corresponding to *M* NAND gates and *M* inverters. The cost of the predecode blocks is negligible.
- *M*×*N* one-transistor cells for memory array.
- Two column MUXes, each consisting of *N* pass transistors and a column decoder consisting of *N* NAND gates, for a total cost of *10N*.
- *W* sense amplifiers/drivers, each with 10 transistors, for a cost of *10W*. Therefore, the total base area of the memory array is *6M* + *MN* + *10N* + *10W* units.

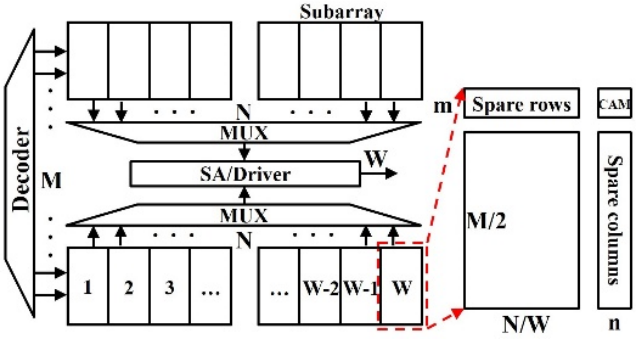

Figure 6: Composition of a mat for redundant repair.

4.1.2. Area models for various schemes

2D redundant repair The overhead of the 2D redundancy technique is primarily related to the cost of the spare elements and the required peripheral circuitry, as illustrated in Figure 2. After the ESP, or any other algorithm replaces the faulty regions by a subset of the available spare rows or columns, the address of the replacement is stored in a content-addressable memory (CAM). When a memory address is applied, the CAM determines whether the address matches a faulty region. If so, the access is redirected to the appropriate spare element. A multiplexer, which is omitted from the figure for simplicity, is used to select from either the main memory array or the spares, based on the output of the CAM block. The area overhead of the scheme corresponds to the spare rows/columns, the additional complexity in the column multiplexer, and the CAM. The *W* bits in each word are interleaved in memory. For each bit, we consider a subarray, magnified in Figure 6, of size (M/2) × (N/W). Since there are 2W such subarrays in a mat, as shown in the figure, the overhead for redundant repair with m spare rows and n spare columns is as follows:

- The spare rows require $m \times (N/W)$ cells per subarray, so that for $2W$ subarrays, the overhead is $2mN$.
- The spare columns require $c(M/2)$ cells per subarray. Over all of the 2W subarrays, the total overhead is nMW.
- The reconfiguration addresses are stored in a CAM of size $\log(M/2)$and $log(N/W)$, respectively, for the rows and the columns. Since each CAM cell consists of 10 transistors and uses a sense amplifier with 4 transistors, the number of transistors in the CAM for each subarray is $T_{CAM} = m \times (10\log(\mathrm{M}/2) + 4) + n \times (10log(N/W) + 4)$.

- The MUXes for the spare columns require n pass transistors, for total of $2nW$ transistors, with an overhead of $2mN + nMW + T_{CAM}(m,n) + 2nW$ units.

Segmented memory repair For the segmented repair model, if the rows and columns are both divided into *P* parts, then the overhead is the same as 2D redundant repair, except that the cost of the CAM is now $P \times T_{CAM}(m,n)$, where $T_{CAM}(m,n)$ is as defined above.

***k*-MSB repair** For *k*-MSB repair, the arrays containing *k* MSBs of the byte use *m* spare rows and *n* spare columns, while other arrays use fewer spare elements, with m_{less} rows and n_{less} columns. These modifications are made to the first two terms of 2D redundant repair, and the overhead of the CAM is $k \times T_{CAM}(m,n) + T_{CAM}(m_{less}, n_{less}) \times (W-k)$.

Bit-shuffling and compressed bit-shuffling The overhead of bit shuffling is based on the hardware scheme in Figure 3. The compressed bit-shuffling scheme is parameterized by n_{FM} and the compression rate R, where R is the number of bytes in a group that share a shift value marker. For an $M \times N$ mat, $M \times N/(8R)$ bytes have their own shift values with $logR$ bits to locate their position. For the structure in Figure 4, the transistor overhead is shown below:

- The shift values for a byte need n_{FM} bits of storage and are shared over R bytes. For M rows and N columns, this results in a total cost of $(M \times N/(8R)) \times n_{FM}$.
- The identity of which of the R bytes should be shifted is stored in a small array with $logR$ bits per group. Its transistor overhead is$(M \times N/(8R)) \times logR$.
- $(n_{FM} + logR) \times N/(8R)$pass transistors are required.
- The 8-bit circular shifter has $n_{FM} + 1$ layers, with 8 MUXes per layer, each with 6 transistors. This leads to a transistor count of $48(n_{FM}+1)$. The total overhead for this scheme is thus $(M \times N/(8R)) \times (n_{FM} + logR) + (n_{FM} + logR) \times N/(8R) + 48(n_{FM}+1)$ units.

Hybrid bit-shuffling and redundant repair The overhead can be deduced from the above discussion, adding the cost of bit-shuffling to the overhead of redundant repair.

4.2 Delay model

The delay model is based on the delay associated with the CAM, the addressing delay, the precharge delay, and the cell read-write time. For the 1Gb DDR3 DRAM in [15], $t_{RCD} = 13.91ns$ and $t_{CL} = 13.91ns$. Thus, the delay to read from memory is $t_{RCD} + t_{CL}$, which includes opening a row-by-row decoder and accessing the memory array and the active column MUX to get the data out. For the 16Mb STT-MRAM in [16], the read cycle time is $35ns$.

All of the redundant repair schemes-2D redundant repair, segmented redundant repair, and *k*-MSB redundant repair- need a CAM to store the reconfiguration address for the spare elements. The matchline delay of the CAM is determined by the precharge and evaluation time [17], where $t_{pre} = 2.2R_{EQpre}C_{ML}$, $t_{eval} = 0.69R_{ML}C_{ML}$, R_{EQpre} is the equivalent resistance of the precharge transistor, R_{ML} is the equivalent resistance in matchline, and C_{ML} is the equivalent capacitance. The delay of the MUX used to select the data from the spare elements or the main memory array can be estimated as three FO4 delays. All of the corresponding parameters for the CAM and FO4 come from CACTI [10] for the DRAMs case and NVsim [18] for the STT-MRAMs to compute maximum timing overhead$(Max_delay_{CAM} < 0.2ns, Max_delay_{MUX} < 0.1ns)$.

For the bit-shuffling related schemes, the chief delay overhead is the circular shifter due to delay in read shift value operation partially overlaps delay in memory read or write operation. The shifter can be implemented using $n_{FM} + 1$ MUX layers. For our case, the maximum value is four MUX layers, giving a total delay of 12 FO4 ($< 0.4ns$).

Based on maximum timing overhead for various parts of the array, it is clear that, compared to the original memory access time, the delay overhead of the additional circuitry is minimal and *can be neglected.*

5. Experimental results

We evaluate the performance of each repair technique for a $128M \times 8$ banks DDR3 DRAM similar to [15] and a $1M \times 16$ bit STT-MRAM similar to [16] using the CACTI memory organization. For each memory type, various faults are injected with the mean failure rate, P_{cell}, set to 0.1%, 0.25%, 0.5%, and 0.75%. These failure injection rates are significantly higher than those typically used in prior memory repair studies since we are evaluating error-resilient applications which can use memories that are much more error-prone and, hence, less expensive. The simulations use 10^5 Monte Carlo samples corresponding to the distribution in Section 2.2.1. Each of these samples corresponds to the total number of faults in a memory array. The probabilities that these faults map on to row faults, column faults, cluster faults, and single cell faults are set to 1%, 10%, 2%, and 87%, respectively [11]. Once the fault region is fixed for a failure, the fault type is set to one of options described in Section 2.2. For purposes of repair, the precise type of fault is immaterial since the existence of a fault would trigger the use of a repair strategy. The impact of the fault on the quality of the JPEG compressed image does not depend on the type of memory, i.e., DRAM or STT-MRAM, but rather on the total number of faults and their spatial distribution.

We use the PSNR to measure the quality of the images stored in the memory arrays as various faults are applied. Our experiments (details omitted due to space limitations) show that the distribution of the PSNR for the Monte Carlo simulations is always very tightly spread about the mean so that the variance is negligible. Therefore, it is sufficient to characterize the PSNR performance results using only the mean values. To compare the different schemes, we define the *quality degradation* (measured in dB) and the *percentage quality degradation* as the change in PSNR compared to the full quality image. We evaluate the different schemes by showing the changes in the quality degradation metric as a function of the area overhead, which is expressed as a percentage of the total area of an equivalent size memory array with no repair capability plus its supporting circuitry.

Figure 7 shows the quality-overhead trade-offs for the bit shuffling schemes. Within each chart, the different bars within a group represent the various cell failure probabilities. The different charts within each column show the results for various compression rate of compressed bit-shuffling scheme. In reality, the memory arrays have row faults, column faults, and cluster faults. In this case, the quality degradation goes up significantly for all the bit-shuffling schemes. Figure 8 shows the quality-overhead trade-offs for the hybrid scheme that combines bit-

shuffling with redundant repair and the term "H-Compression rate" represents the hybrid scheme with compressed bit-shuffling scheme. The four curves correspond to various cell failure probabilities and the points on a curve represent $n_{FM} = 1, 2, 3$. For the same n_{FM}, the number of spare elements added for redundant repair vary for the different failure probabilities, resulting in different area overheads. As compression rate increases, the overhead will be reduced with minimal quality degradation. The hybrid schemes significantly reduce the quality degradation for cluster faults with only a small increase in overhead compared to only bit-shuffling schemes. Thus, the following comparisons only include the hybrid schemes.

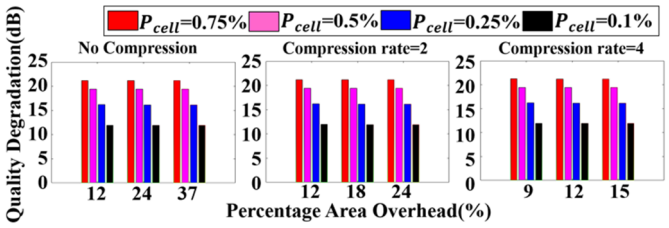

Figure 7: Quality-overhead trade-offs for bit-shuffling with clustered faults and the points on the x-axis correspond to $n_{FM} = 1, 2, 3$.

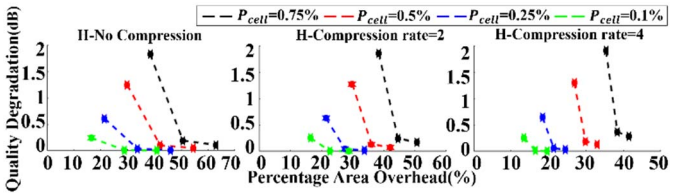

Figure 8: Quality-overhead trade-offs for the hybrid scheme with clustered faults.

Figure 9 compares all approximate memory repair schemes. To provide another perspective, this figure exchanges the horizontal and vertical axes used in prior graphs. The x-axis groups various ranges of quality degradation, expressed as a percentage increase in degradation compared to the perfect memory. The y-axis shows the area overhead. Each plots shows the results for a different fault probability. These plots allow a designer to determine the cost of a specific scheme given a maximum allowable quality degradation or, given a cost budget, to determine the expected quality degradation of each scheme.

To evaluate the schemes that use limited spare rows and columns, we first choose the number of spare rows and columns that are required to ensure full repair over all of the Monte Carlo samples for 2D redundant repair and segmented repair. This value then is adjusted from 100% for a fully repaired system to various reduced values, each of which corresponds to reduced overhead at the cost of some quality degradation. Figure 9 shows the level of quality degradation for various schemes and the corresponding overhead. In our simulation, we segment the memory into 2, 4 and 8 blocks and the results show that the segmented repair scheme works best with only two blocks. This is because, as the number of segments increases, the number of spare cells goes down, but the number of high cost CAM cells will increase. However, both the 2D and segmented repair schemes always have the highest overhead for a given level of quality degradation since these schemes are not very efficient.

It is clear that the performance of the *k*-MSB scheme is substantially better than the 2D and segmented repair schemes providing lower overhead and lower quality degradation, even for *k*=1. Furthermore, decreasing *k* tends to reduce the overhead for a given level of quality degradation by improving the efficiency, but lower values of *k* may be unable to achieve the best quality. The hybrid bit-shuffling with redundant repair scheme shows the best performance as the probability of a fault increases. However, because every byte needs extra space to store the shift value, the hybrid scheme has relatively high overhead compared to the other schemes when the probability of a failure is low. Moreover, the hybrid scheme with compression can reduce the area overhead with lower quality degradation compared to no compression. In summary, we find that the *k*-MSB and hybrid schemes generally provide the best results across all failure probabilities.

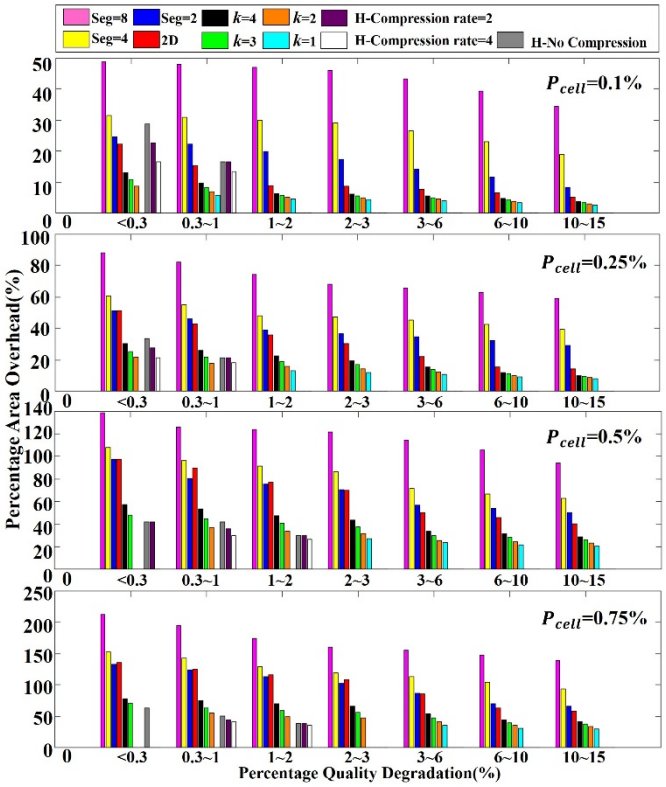

Figure 9: Overhead-quality comparisons of the various approximate memory repair schemes for different failure probabilities. Missing bars (e.g. *k*=1 for $< 0.3\%$ degradation under $P_{cell} = 0.1\%$) show configurations that are impossible to actually build for the given overhead-degradation combination.

6. Conclusion

This work has shown how partially repaired memories can be effectively used for error-resilient image applications. We find that the nature of the fault and the type of memory matters less than the number and spatial distribution of the faults. For image compression, the proposed *k*-MSB and hybrid bit-shuffling and redundant repair schemes provide large reductions in overhead compared to the fully-repaired case and prior repair schemes with little quality loss. Our new approaches perform well in the presence of row, column, and

clustered faults, where the previously proposed basic bit-shuffling method may experience large quality degradation.

7. Acknowledgement

This work was supported in part by C-SPIN, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP), a Semiconductor Research Corporation program sponsored by MARCO and DARPA; and by National Science Foundation grant no. CCF-1438286. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

8. References

[1] B. Schroeder, E. Pinheiro, W. D. Weber, "DRAM errors in the wild: a large-scale field study," ACM Sigmetrics Performance Evaluation Review, vol. 37, no. 1, pp. 193-204, 2009.

[2] C. Slayman, "Soft error trends and mitigation techniques in memory devices," Reliability and Maintainability Symposium (RAMS), pp. 1-5. 2011.

[3] J.-F Li, J.-C Yeh, R.-F Huang, C.-W Wu, "A built-in self-repair design for RAMs with 2-D redundancy," IEEE Transactions on VLSI Systems, vol. 13, no. 6, pp. 742-745, 2005.

[4] S.-K Lu, C.-L Yang, Y.-C Hsiao, C.-W Wu, "Efficient BISR techniques for embedded memories considering cluster faults," IEEE Transactions on VLSI Systems, vol. 18, no. 2, pp. 184-193, 2010.

[5] Y. Xie, *Emerging memory technologies*, Springer, 2014.

[6] I. Koren and Z. Koren, "Defect tolerance in VLSI circuits: techniques and yield analysis," Proceedings of the IEEE, vol. 86, no. 9, pp. 1819-1838, 1998.

[7] A. Sampson, J. Nelson, K. Strauss, L. Ceze, "Approximate storage in solid-state memories," ACM Transactions on Computer Systems (TOCS), vol. 32, no. 3, 2014.

[8] S. Ganapathy, G. Karakonstrantis, A. Teman, A. Burg, "Mitigating the impact of faults in unreliable memories for error-resilient applications," ACM/EDAC/IEEE Design Automation Conference, 2015.

[9] M. Horiguchi and K. Itoh, *Nanoscale memory repair*, Springer Science & Business Media, 2011.

[10] "CACTI tools", http://www.hpl.hp.com/research/cacti/.

[11] T.-H Wu, P.-Y Chen, M. Lee, B.-Y Lin, C.-W Wu, C.-H Tien, "A memory yield improvement scheme combining built-in self-repair and error correction codes," IEEE International Test Conference, pp. 1-9, 2012.

[12] C.-T Huang, J.-R Huang, C.-F Wu, C.-W Wu, T.-Y Chang, "A programmable BIST core for embedded DRAM," IEEE Design & Test of Computers, vol. 16, no. 1, pp. 59-70, 1999.

[13] A. J. Van de Goor, *Testing semiconductor memories: theory and practice*, John Wiley & Sons, Inc., 1991.

[14] S.-K Lu, Y.-C Tsai, C.-H Hsu, K.-H Wang, C.-W Wu, "Efficient built-in redundancy analysis for embedded memories with 2-D redundancy," IEEE Transactions on VLSI Systems, vol. 14, no. 1, pp. 34-42, 2006.

[15] "DDR3-SDRAM data sheet". https://www.micron.com/products/dram/ddr3-sdram/.

[16] "STT-MRAM data sheet", http://www.everspin.com/.

[17] K. Pagiamtzis, A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," IEEE Journal of Solid-State Circuits, vol. 41, no. 3, pp.712-727, 2006.

[18] "NVsim tools", www.nvsim.org/.