A Pageable, Defect-Tolerant Nanoscale Memory System

Susmit Biswas, Tzvetan S. Metodi*, Frederic T. Chong, Ryan Kastner
University of California, Santa Barbara
University of California, Davis*
{susmit,chong}@cs.ucsb.edu, kastner@ece.ucsb.edu, tsmetodiev@ucdavis.edu

***Abstract*—As we scale down to the nanoscale regime, manufacturing defects will increase significantly. With expected bit error rates as high as 2-10%, the reliability of a contiguous 4 K-Byte memory page falls off to zero. We propose a powerful combination of static and dynamic techniques to tackle this problem. Using a combination of defect mapping, error correction, and sparing, we can achieve approximately 46.5%, 26.1% and 13.2% storage efficiency in contiguous 4 K-byte pages, given bit error rates of 2%, 5% and 10%, respectively. This result allows us to use standard virtual memory to map a contiguous virtual address space onto a nanoscale memory system with some bad physical pages, giving us a more usable system than previous approaches.**

I. Introduction

The success of our computing industry has been substantially fueled by the rapidly diminishing size of devices. As this device scaling reaches the nanometer regime, however, we are faced with either large fabrication costs or high defect rates. We propose a powerful combination of static defect mapping, dynamic error correction codes, and sparing to produce a robust memory system from highly-defective bits. In particular, we can produce more than 46 percent storage efficiency in 4K-byte pages from a defect rate of 2 percent per bit. This result allows us to use standard virtual memory to map a contiguous virtual address space onto a nanoscale memory system with some bad physical pages, giving us a much more usable system then previous approaches that produce non-contiguous memory [1], [2], [3], [4].

The motivation for our work is based on the observation that, building a virtual memory system using non-reliable memory having even a 2% bit error rate will result in virtually zero (order of 10^{-180}) usable 4K-Byte pages unless robust error correcting codes are used. Moreover, the predicted error rate in future nanoscale memory devices is expected to be much higher than 2% and reach as much as 10% [1]. To create an efficient, pageable nanoscale memory, we propose a hierarchical scheme. Using strong error correcting codes at the block level, spare memory blocks, and encoding the defect map bits with the encoded data bits, we can achieve page-level storage efficiency as high as 13.2% at the cost of added decoding overhead and minimal reliable memory requirements of only 3.05×10^{-3} percent of the non-reliable memory (in terms of bits). To our best knowledge, this system outperforms previous approaches in building reliable and usable memory from defective nanoscale devices by several orders of magnitude.

Note that our results are orders of magnitude better than expected, as the reliability of 32K of contiguous bits drops off exponentially with each bit. To overcome this challenge, we use a combination of defect mapping and sparing. Since defects can be located during manufacturing time, a static defect map can be created that can exploit significantly more information than dynamic techniques. Storing the defect maps, however, can be expensive when defects are mapped on a per bit basis. Consequently, we use dynamic error-correcting codes (ECC) to bootstrap our bit level reliability to 32-bit memory blocks. In other words, we take a bit defect rate of 10 percent create 32-bit block such that it reaches a moderate defect rate of 0.01 percent. We then use our defect mapping and spare-block techniques to leverage the blocks into 4K-byte pages with memory efficiency of more than 10 percent. Note that ECC by itself can not address the high defect rates that we target. To achieve high defect tolerance using only ECC results in very high overhead in area and delay. It is only the combination of static and dynamic approaches that is sufficient.

Finally, it is critical to provide a contiguous address space for our system to be usable. We use two mechanisms. First, we provide spare blocks to fill in for bad blocks in each 4 K-byte pages. Second, we use the virtual memory system to map out any pages that still have bad blocks after sparing.

The rest of the paper is organized as follows. In Section II, we give a description of the related research work. In Section III, we present an overview of our system model and assumptions. Our methodology and results are discussed in Sections IV and V respectively. Finally we summarize our results in Section V and conclude with a perspective of future work in Section VII.

II. Related Work

Currently manufactured CMOS and disk technologies have error rates in the order of 10^{-9} and have not required rigorous error correcting codes and mechanisms. It is widely expected, however, that memories built in deep submicron technologies will have manufacturing defect rates as high as 10% [5], [6], [7]. In the context of nanoscale memory, we adopt a first-order model of these defects in terms of bit-level errors (we discuss extensions to this model in our future work). To address high

bit-error rates, a wide variety of research exists analyzing the use of error correcting codes to encode and protect the data using the concept of redundancy. Jeffery et al. [8] have proposed a three-level error correcting memory architecture for nanoscale devices using single or double error correcting codes, 4^{th} level RAID and sparing of RAID arrays. Han *et al.* [9] apply NAND multiplexing techniques, based on a high degree of redundancy (greater than 1000) of imperfect devices and interconnect components to achieve high reliability in nanoscale fabrics. Built-In-Self-Repair (BISR) based architectures [10], [11], [12] for defective memories have been proposed which use a high degree of redundancy in memory rows and columns.

Ou *et al.* [13] propose hardware design for the decoding and encoding routines of Hamming error correcting codes [14], where the memory reliability is increased at the cost of $5ns$ delay in the memory access time. Their scheme, however, works best for error rates lower than 10^{-6}. For high error rates, stronger codes such as BCH codes [14] are required and have been investigated by Sun *et al.* [3]. One problem, however, is that, even if strong error correcting codes can significantly improve the reliability of each bit, the yield at the memory byte level is still very low and even lower at the block and page levels. For example, the BCH (250,32,45) code encodes 32 logical bits into 250 physical bits and can correct as many as 45 errors on the physical bits and provides logical block success rate up to 99.9956% at a 10% bit-error rate. Yet 1 out of every 711 bytes is expected to be defective. Stronger codes exist, but complex decoding schemes may introduce enough overhead as to neutralize the density and speed advantages of nanoscale device memories.

One possible solution, is to use use defect maps to store the locations of the defective bits [15]. For reconfigurable architectures, tile based memory units have been proposed where components store the defect map in a distributed fashion [1], [2], [16]. The drawback, however, of bit-level defect maps is that the storage overhead is usually very high. Tahoori [17] proposes a *defect unaware design flow* which identifies universal defect free subsets within the partially defective chips to reduce the size of the defect map. Wang *et al.* [4] propose the use of Bloom filters [18] for storing defect maps for nanoscale devices. Bloom filters, however, require frequent hashing, which is is computationally expensive and may significantly increase the memory access times. The authors in [3] propose the use of employing CMOS memory for storing metadata to identify defect-free portions of the memory using two schemes: (i) a two-level hierarchy of CMOS and nano-device memory; (ii) a *bootstrapping* technique to store the reliable block information in some defect-free portion of the non-reliable memory and storing this index in the reliable CMOS. The problem, however, is that the amount of memory to store the ranges increases with the sparseness of faulty memory bits. Moreover, to build a memory compliant with commodity virtual memory systems, we need to have fixed-size pages instead of variable-size ranges.

The most important contribution of this work is the development of a design for a reliable, large-scale pageable memory at bit-error rates suitable for expected nanotechnology devices: something that has not been addressed in prior work. We find that it is difficult to allocate memory efficiently using list-based defect maps, and discover that we can achieve large page yields if we use spare blocks to replace defective memory blocks. Moreover, we encode the mapping of spare blocks (reconfiguration bits) *with* the data bits into the same error correcting code. We do this joint encoding without increasing the number of physical bits used. For example, we may encode 32 data bits with 2 reconfiguration bits in 250 physical bits, the same number of physical bits we would have used to encode the 32 data bits alone. This allows us to trade storage overhead with a small decrease in defect tolerance.

III. Defect Model

Many nanoscale technologies are still in their infancy and the nature of manufacturing defects in these technologies is not well understood. There is, however, reasonable consensus that defects will be a serious challenge, especially when trying to keep manufacturing costs under control. Our study uses a defect model of uncorrelated bit errors as, to our best knowledge, the accurate correlation in manufacturing defects is not established. In this section, we examine some technologies and discuss some of the issues relating to this first-order model.

In order to decrease the area of a memory device, either the size of the memory cell or the spacing and pitch of interconnects need to be reduced. Scaling beyond that $32nm$ point results in several problems, including electromigration, increased capacitance and resistance, and power. Therefore, the research community has started looking into different interconnect technologies, among which CNT based interconnect[19] is very promising as it is less susceptible to electromigration and demonstrates better characteristics in terms of delay and power compared to traditional metal-interconnects. CNT devices are still in the investigation stage of development and good defect models do not yet exist. In our analysis, we use the intuition that defects in local wires will result in random defects, whereas defects in global wires will result in clustered defects. We also make the assumption that global wires can take more area and be made using reliable technology. For this initial study, we assume that many of the local defects in the memory interconnect will be randomly distributed, and thus pose a much bigger problem than correlated errors that can be routed around with clever routing algorithms [11].

Our design approach is not targeted towards any particular deep submicron technology. Instead, we addresses reliability issues commonly expected for all nanoscale technologies such as Molecular Memories, Polymer Memories, Carbon Nanotubes (CNT) and many others described in the ITRS technology roadmap [5]. We assume that manufacturing defects can be recorded during the testing phase of nanoscale memories. In fact, there has been ample research work in testing memory for manufacturing defects [20], [21]. Most prominently, Mishra et al. [7] propose a scalable testing

solution which can be used for nanoscale fabrics using a *two phase* algorithm where components with high probability of being defective are identified in the first phase and low probability components are tested in the second phase using tile-based test patterns. Therefore, we assume that testing for defects is feasible and the defective bits can be known prior to computation and memory architecture design. Once the defects are identified, our system is able to produce satisfactory reliability from the non-reliable memory through the system design techniques we evaluate in this paper.

IV. Methodology

In our spare block scheme, spare blocks are used as a replacement for defective 32-bit logical memory blocks and the b-bit reconfiguration information (for the mapping of spares to bad blocks) is stored adjacent to the data bits in each $(32+b)$-bit block. The overview of the memory architecture is shown in Figure 1. Data and metadata are entangled in encoded form and stored in memory blocks. Upon decoding, the metadata and data components are separated. Metadata consists of *defect map* and *spare map* information as shown in Figure 1. As the metadata is stored in unreliable blocks, majority voting is employed to ensure the correctness of metadata. In other words, the defect map metadata uses one bit for every block of data participating in voting indicating the usability of the block. Spare map or reconfiguration information stores the mapping of the defective blocks to the spare blocks. We explain the function of spare map in the later part of this section in more detail. Reordering logic permutes the data blocks substituting the defective blocks with spare blocks.

We use strong error correcting codes to reduce the error rate of a block of bits to manageable levels given bit error rates as high as 10%, where the reconfiguration metadata bits are encoded *with* the data bits in the same error correcting code block. One of the biggest drawbacks in the existing literature, is that the reconfiguration bits must be deterministically reliable in order to avoid writing data to faulty memory locations. Thus building reliable memory devices over unreliable nanoscale devices reduces the density advantage of the nanoscale memory. The key to our design is that, by encoding the reconfiguration bits in the same ECC code block of the 32 logical data bits required for each memory block, we can create deterministically reliable reconfiguration bits using unreliable memory components.

Consider, for example, a memory page which keeps two spare blocks for every sixteen 32-bit data blocks. The spare blocks can be used to replace any faulty page block. Given that we only have two spare blocks, the entire page is thrown away if more than two of the sixteen blocks are faulty. Defect map bits that store the information about block defects are encoded with 32-bit data blocks using the same error correcting code and by applying a majority voting scheme involving 5-blocks, we ensure that we identify a faulty block as usable only when more than two blocks have failed. However, if more than two blocks have failed, the entire page is discarded by the memory system. If less than or equal to two blocks fail in a set of sixteen blocks, majority voting scheme discovers the faulty blocks. Map of the spare blocks is obtained from the good blocks, and then spare blocks are used to replace the defective blocks. It is possible for a spare block to be defective, but the total number of bad blocks in the set of eighteen blocks cannot be more than two.

To give us needed flexibility for customized encoded block lengths (32 bits plus reconfiguration bits), we choose the BCH error correcting code [14]. In particular, a BCH (n, l, t) code that encodes l logical bits into n physical bits and can correct t errors, can be converted to a BCH $(n-s, l-s, t)$ code without major modification in decoding procedure, where s is an integer such that $s < l$. The decoding procedure for the BCH code consists of three stages: (i) Syndrome decoding; (ii) Calculating error locator polynomial; (iii) Obtaining roots of the polynomial to find the error locations; The decoding procedure can be implemented using a bit parallel technique to reduce its delay [22]. We describe the overhead of using the BCH code in more detail at the end of this section.

By encoding the reconfiguration bits with the data bits, we dilute the efficiency of the error correcting code and decrease the reliability of our data bits. We are taking a BCH code that encodes 32 logical bits into n physical bits and converting it to a $32+s$ logical bits into $n+s$ physical bits, however it can still correct the same number of errors (s is the number of reconfiguration bits). In essence, we trade some of the data reliability to improve the area and performance of our memory model. This is a winning trade, however, since we observe only an insignificant decrease in the block reliability, as shown in Table I.

TABLE I
EFFECT OF ASSOCIATING 2 RECONFIGURATION BITS WITH DATA. THE ADDED *defect map* BITS DECREASES THE BLOCK RELIABILITY AND INFORMATION RATE OF THE BLOCK BY SMALL AMOUNT, BUT NULLIFIES THE NEED OF STORING DEFECT MAP IN RELIABLE MEMORY. THE SPATIAL LOCALITY OF DEFECT MAP WITH DATA BITS REDUCES THE OVERHEAD OF FETCHING DEFECT MAP FROM A DIFFERENT MEMORY LOCATION.

Encoded	Code (BCH)	Block Yield (%)	Information Rate
Data	224,32,30	95.9751	0.1429
Data and Metadata	226,34,30	95.5455	0.1416

In the following analysis, we assume that the bit error rate in the unreliable memory is r_b and the length of the logical data block is l_B bits. Logical bits are encoded using a BCH (n, l, t) code and b number of spare blocks are allocated for every B number of blocks. The memory is partitioned into pages each of which has S number of blocks. In order to correct up to b bad blocks, $2b+1$ defect map bits are required, which will be able to correctly identify the faulty blocks through majority voting. In a simple scheme, we might store the reconfiguration map in every block for all blocks in a set i.e. $(2b+1)\lceil log_2(b+1)\rceil$ bits in every block. However, it can be minimized by keeping spare block address of b blocks. As there can be at most b bad blocks in a set of $2b+1$ blocks, it is sufficient to store the address of b spare blocks which can

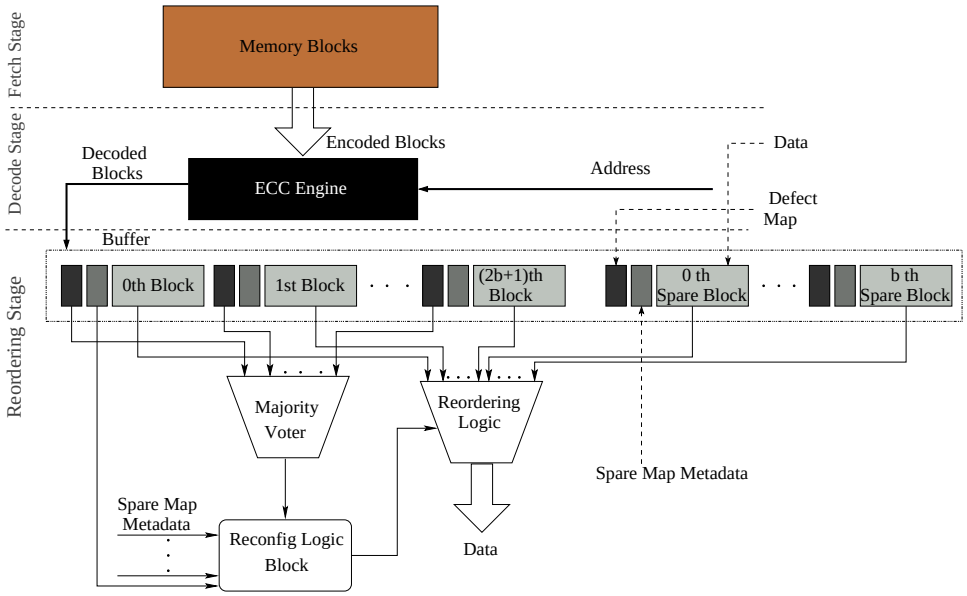


Fig. 1. Overview of decoding scheme. The process is pipelined into three stages where the 1^{st} stage fetches encoded blocks from unreliable memory. 2^{nd} stage is used to decode the data and 3^{rd} stage pushes the data in buffer. When the buffer is filled, voter discovers the bad block from the set of $2b+1$ blocks. If the number of bad blocks in $2b+1$ blocks and b spare blocks is less than b, then defectless data can be read. The number of stages in the pipeline can be further increased by breaking the decoding segment. The buffer, decoder, voter and reordering unit are built using reliable technology which can be placed outside the nanoscale memory.

replace the defective blocks. Therefore, every block of data has to be encoded with $2b+1+b\lceil log_2 b\rceil$ metadata bits. The spare map bits ($b\lceil log_2 b\rceil$) can be excluded from the metadata if all the B blocks are read out for majority voting, however, we use it to reduce the amount of block reads. Using this scheme, to read one block of data, we need to read $2b+1$ blocks, whereas if the spare map is not encoded with data, it will be necessary to read B blocks at a time.

Therefore, the $BCH(n,l,t)$ code has to be converted into a $BCH(n+l_B+2b+1+b\lceil log_2 b\rceil - l, l_B, t)$ code [14] for storing reconfiguration bits with data bits. The new block error rate can be formulated as,

$$\begin{aligned} r_B &= 1-\sum_{i=0}^{t} C_i^{n+l_B+b\lceil log_2 b\rceil+2b+1-l} \\ &\quad r_b^i(1-r_b)^{n+l_B+b\lceil log_2 b\rceil+2b+1-l-i}, \end{aligned}$$

where C_i^b is number of possible combination of i items from a set of b items and the there are s number of b sized blocks. For a set of blocks of length $B+b$, the probability of having equal to or less than b bad blocks is:

$$r_{set} = \sum_{i=0}^{b} C_i^{B+b} r_B^i (1-r_B)^{B+b-i} \tag{1}$$

Therefore, yield of a memory page, *i.e.* the probability of the entire memory page being usable is:

$$r_{sys} = (\sum_{i=0}^{b} C_i^{B+b} r_B^i (1-r_B)^{B+b-i})^{S/(B+b)} \tag{2}$$

However, the memory yield metric does not reflect the overheads of sparing and error correction. Considering the overhead of spare blocks and error correcting redundancy, the memory efficiency can be defined as

$$\begin{aligned} E_{mem} &= (B/(B+b)) \\ &\quad \times (l_B/(n+l_B+b\lceil log_2 b\rceil+2b+1-l)) \\ &\quad \times \sum_{i=0}^{b} C_i^{B+b} r_B^i (1-r_B)^{B+b-i})^{S/(B+b)} \end{aligned}$$

Instead of having fixed sized pages with each having S number of blocks, we can design a system with variable page sizes where the page sizes are aligned to *2-power-boundary*. In that case, the probability of allocating an s sized page from S blocks having b extra blocks per B blocks is

$$Prob(s_i, S) = r_{set}^{S/B} - \sum_{s_j=2s_i}^{s_{j+1}=2s_j, S} Prob(s_j, S), \tag{3}$$

, where r_{set} is the yield of a set of B blocks.

A. Complexity of BCH code decoding

For a *bit-parallel* implementation of $BCH(n,k,t)$ code [22], the decoding time in terms of $FO4$ CMOS delay, and the area in terms of F^2_{CMOS} are $\tau_{BCH} = O(t(log_2 n + log_2 t))$ and $A_{BCH} = O(nt(log_2 n)^2)$ respectively. The area and delay, is computed using 45 nm technology and our results are summarized in Table II. For 22 nm technology, the values are smaller by an order of magnitude. For stronger BCH codes, the area and delay overhead is much worse e.g. BCH(511, 103, 61) code has a decoding delay of $130ns$ and area of $1mm^2$ though the block level fault tolerance is increased

to 99.6093% from 99.5691% which can be obtained using $BCH(255, 37, 45)$ code. Therefore, we restrict ourselves to the limit of $BCH(255, 37, 45)$ code.

V. Results

Error correcting codes provide reliability to a certain degree of error rate beyond which the performance decreases as shown in Figure 2. We compute the yield in terms of pages while varying the number of spare blocks per 8, 64 and 1024 blocks in a set. It can be observed from the figure that the spare-block-based scheme improves the defect tolerance by an order of magnitude in BCH code over $GF(2^8)$. It is observable from the figure that the curve corresponding to our scheme is offset by $1.6 - 3\%$ from the case with no sparing. In other words, memory built using spare block based scheme is able to tolerate $1.6 - 3\%$ more errors than using error correction only, which is significant in our range of 2-10% bit error rate.

Supporting variable-sized pages can have a large impact on memory yield. Witchel et al. proposed [23] Mondrian Memory protection system which uses variable-sized pages instead of fixed sized pages. This, consequently, increases the yield of the memory, but the metadata management and the amount of metadata increases to a large extent. In Figure 3, we show the availability of variable-sized blocks. We speculate that variable page size might become more popular in future technologies where manufacturing defects are a significant problem.

Optimizing all the factors affecting the yield of our memory system, we obtain the best configuration for all of the error correcting codes, and we can observe from Figure 4 that the best is obtained with a strong $BCH(255, 45, 43)$ code which results in 13.2% memory efficiency with the overhead of all redundancy including error correcting code and sparing redundant blocks. For high error rates, our scheme outperforms the base scheme by orders of magnitude. Even if the strongest BCH(255,37,45) code is employed, our scheme results in 0.4% memory efficiency enhancement, whereas the difference is larger in case of weaker codes. In short, the optimal spare block based scheme results in graceful degradation of performance compared to the case of using error correcting code only.

A. Comparative Study

We compare the scheme with two prior schemes, namely the list-based approach by Sun et al. [3] and Bloom filter based approach by Wang et al[4]. For the list-based approach, the amount of metadata to be stored in reliable CMOS is $\approx 3 \times 10^2/10^5$ i.e. 0.3% of the unreliable memory (measured in terms of bits, but reliable bits may be significantly larger than unreliable bits). Using a Bloom filter to store metadata at bit-level requires 60% of the unreliable memory with 5.6% false positive rate . Using our technique, the metadata is hierarchically distributed in unreliable and reliable memory, but it requires less than $2^{-15} \approx 3.05 \times 10^{-3}\%$ reliable memory.

1) Area: As observed from ITRS roadmap, metal interconnect pitch is the main bottleneck in shrinking memory size. According to the roadmap, less than $96nm$ pitch global metal wires are not manufacturable. Moreover, at lower pitch, physical phenomena *viz.* self-heating, electromigration [24] are severe problems in metal interconnects. Using CNT interconnects, deep submicron ($\leq 10nm$) density can be achieved. However, metal wires can be used for global wiring for better reliability. As, the routing is done not in the same layer as memory cells, area of the memory is not constrained by the interconnect pitch.

In the best configuration, the storage efficiency of the memory is $\approx 13.2\%$ for 10% bit error rate. If we use $l_1$nm reliable technology with $\approx 100\%$ reliability and $l_2$nm unreliable nanoscale memory. We use a bit-vector of reliable memory indicating the usability of a page. The amount of this reliable memory is 2^{-15} times the un-reliable memory (assuming same page size of $4K - Bytes$ for both the memories). The nanoscale system area requirement for S size memory is $C \times (0.13^{-1} \times l_2^2 + 2^{-15} \times l_1^2)$. For same sized reliable memory, the area requirement is $C \times l_1^2$. Nanoscale memory is beneficial in terms of area if $C \times (0.13^{-1} \times l_2^2 + 2^{-15} \times l_1^2) < C \times l_1^2$. i.e. feature size of nanoscale memory is smaller than 34.64% of the reliable memory. Compared to minimum size reliable cell of $128nm$ width, nanoscale memory having feature size as $10nm$ promises $0.13 * (128nm/10nm)^2 \approx 15\times$ area reduction of memory. Other components in the memory system can be built using unreliable technology, and using error correcting scheme they can be made reliable[6]. In Figure 5, we show the design areas in the spectrum of bit error rates. The shaded area in the figure indicates the ratio of *CMOS device pitch/Nanoscale device pitch* which is effective for reducing memory area.

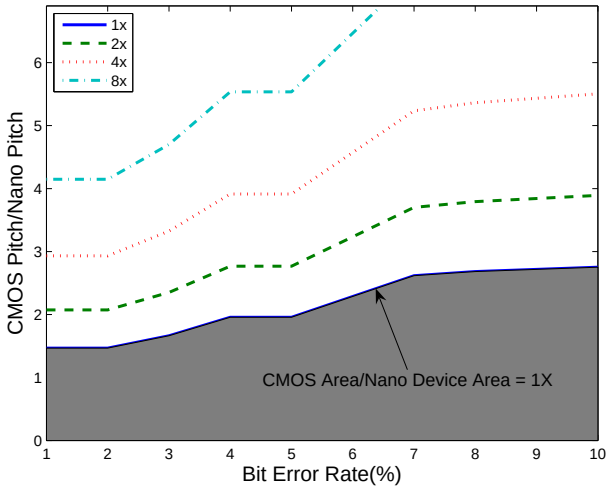
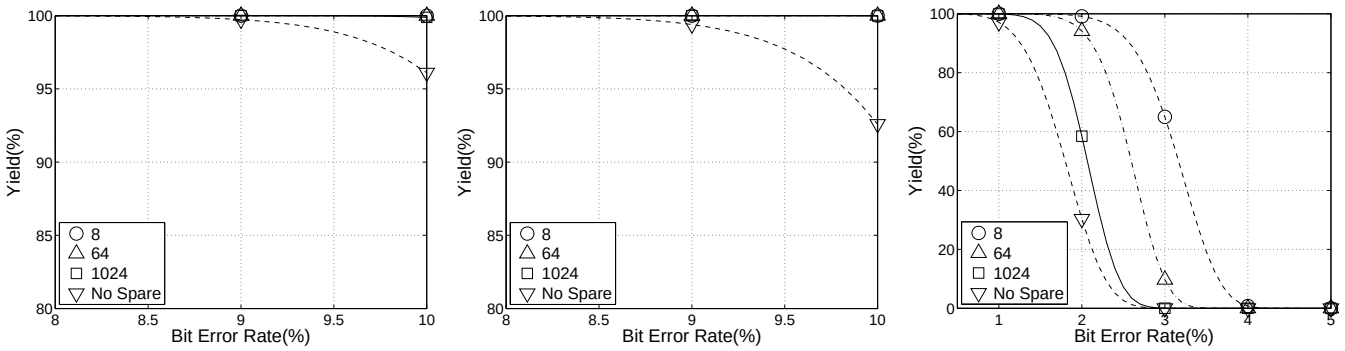


Fig. 5. The design decision for selecting nanoscale device over CMOS. In the x-axis we show the bit error rate varying from 1 to 10%. In the Y-axis we show the ratio of feature size necessary to have smaller size memory. The result is based on the assumption on scaling all the components of memory. It can be observed that at high bit error rate, area benefit requires larger scaling ratio

TABLE II

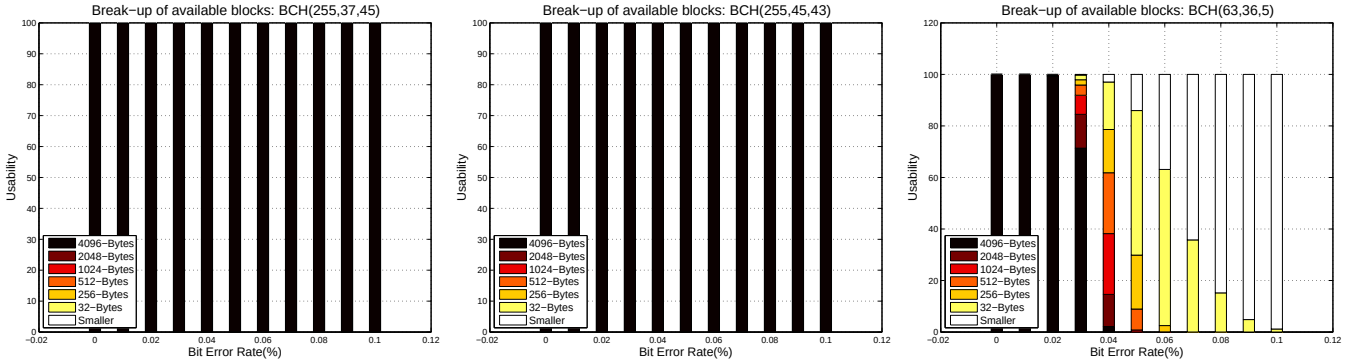
OVERHEAD OF 6 DIFFERENT BCH DECODERS IN TERMS OF DELAY AND AREA. WE ASSUMED A RELIABLE TECHNOLOGY OF $45nm$. THE RESULTS ARE SUMMARIZED FROM THE ONES PRESENTED BY LIKHAREV ET AL.[22]

BCH Code	255,37,45	255,45,43	255,55,31	255,99,24	127,43,14	63,36,5
Area (mm^2)	0.33	0.3	0.27	0.2	0.05	0.0067
Delay(ns)	62.5	60	45	37.5	17.5	6



(a) BCH(255-37-45) (b) BCH(255-45-43) (c) BCH(63-36-5)

Fig. 2. Yield in terms of 4K-Byte pages for various degree of reconfigurability. For different error correcting codes, we compare the performance of having optimal number of spare blocks per 8, 64 and 1024 blocks of data in a page with the case of no sparing. Sparing improves the page level yield in all the cases over the scheme of having no spare blocks



(a) BCH(255-37-45) (b) BCH(255-45-43) (c) BCH(63-36-5)

Fig. 3. Yield in terms of maximum size of contiguous regions, given 1 spare block per 8 blocks. We measure the availability of 32B, 256B, 512B, 1K-Byte, 2K-Byte and 4K-Byte pages

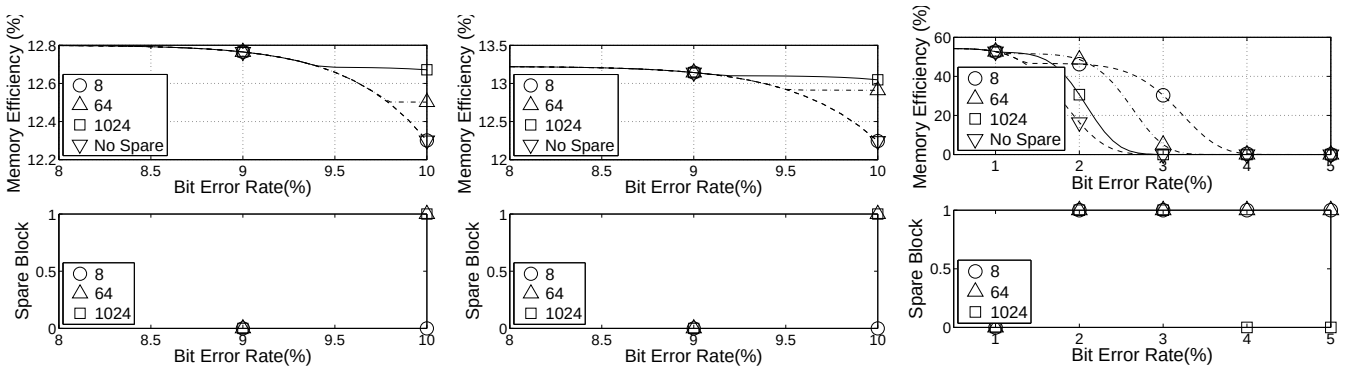
TABLE III

COMPARISON OF STORAGE EFFICIENCY AND USE OF RELIABLE MEMORY IN OUR APPROACH WITH THE LIST-BASED SCHEME PROPOSED BY SUN ET AL. [3] AND BLOOM FILTER BASED SCHEME BY WANG ET AL. [4].

Bit Error Rate(%)	Storage Efficiency(%)			Rel. Conf. Data (%)		
	Our	List	Bloom	Our	List	Bloom
2	46.5	1.5	3.13×10^{-286}	$3.05 \times 10{-}3$	1.154	20
5	26.1	4.99×10^{-3}	≈ 0	3.05×10^{-3}	6.92×10^{-1}	40
10	13.2	9.85×10^{-12}	≈ 0	3.05×10^{-3}	7.70×10^{-2}	60

VI. CONCLUSION

With the scaling of technology, the high bit error rate in memories will make supporting a contiguous address space a serious challenge. In this paper, we show that using a combination of static and dynamic techniques it will be



(a) BCH(255-37-45) (b) BCH(255-45-43) (c) BCH(63-36-5)

Fig. 4. Optimal memory efficiency for each error correcting code. In each plot, the upper subplot shows the memory efficiency and the lower one shows the required number of spare blocks corresponding to each ECC. The optimal spare block based scheme results in significantly better memory efficiency compared to no spare block case.

possible to develop a contiguous and paged memory system from unreliable nanoscale memory devices. We provide an analytical model of memory efficiency in this paper for obtaining configuration of error correcting system. We use ECC to bootstrap our bit level reliability to blocks of 32 bits.We then use our defect mapping and sparing techniques to leverage the blocks into 4K-byte pages with a yield of more than 10 percent. With proper selection of error correcting code and sparing scheme, we can achieve as high as 46.5% and 13.2% storage efficiency in presence of 2% and 10% bit error rate respectively. We also observe that supporting variable page sizes (eg. the Mondrian System [23]) can give us a system that is more resilient to manufacturing defects at the cost of higher metadata and computation overhead. To our best knowledge, our technique promises higher memory efficiency with low requirement of reliable memory as configuration data than any previous approach which aims at building usable memory from defect-prone memory cells.

VII. Future Work

Our first-order defect model focuses on uncorrelated bit errors in the memory system. We plan to explore designs that can better accommodate correlated errors due to physical effects and due to interconnect defects. Non-uniform error distributions may benefit from multiple error correction codes in our memory system. Error correcting decoders and encoders are targeted towards specific error correcting code. However, for lower error rates in some region of memory, memory efficiency can be enhanced by using a weaker code. We plan to explore decoder and encoder designs that could be configured to handle different error correcting codes.

As observed from the ITRS report, scaling down the memory area will require higher-pitch interconnect. A CNT-based interconnect is a prospective candidate for such systems, which can provide better resilience to electromigration and the self-heating effect. However, the mean free path, which is a crucial factor behind maximum interconnect length, needs to be higher than the state of the art technology in order to reduce overhead of buffers and repeaters. Redundancy in the network wiring, buffers and repeaters may be necessary to increase reliability. We plan to explore better layout methodologies that focus more on interconnect scaling.

References

[1] A. DeHon and K. K. Likharev, "Hybrid CMOS/Nanoelectronic Digital Circuits: Devices, Architectures, and Design Automation," in *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, 2005, pp. 375–382.

[2] C. He, M. Jacome, and G. de Veciana, "Scalable Defect Mapping and Configuration of Memory-Based Nanofabrics," in *IEEE International High- Level Design, Validation and Test Workshop (HLDVT)*, 2005.

[3] F. Sun and T. Zhang, "Two Fault Tolerance Design Approaches for Hybrid CMOS/Nanodevice Digital Memories," in *IEEE International Workshop on Defect and Fault Tolerant Nanoscale Architectures (Nanoarch)*, 2006.

[4] G. Wang, W. Gong, and R. Kastner, "Defect-Tolerant Nanocomputing Using Bloom Filters," in *ICCAD 2006*, November 2006.

[5] ITRS, *International Technology Roadmap For Semiconductors - 2006 Edition*. Semiconductor Industry Association, 2006. [Online]. Available: http://www.itrs.net/Links/2006Update/2006UpdateFinal.htm

[6] P. J. Kuekes, W. Robinett, G. Seroussi, and R. S. Williams, "Defect-Tolerant Interconnect to Nanoelectronic Circuits: Internally Redundant Demultiplexers Based on Error-Correcting Codes," *Journal of Nanotechnology*, vol. 16, pp. 869–882, June 2005.

[7] M. Mishra and S. Goldstein, "Defect Tolerance at the End of the Roadmap," in *ITC*, 2003, pp. 1201–1211. [Online]. Available: citeseer.ist.psu.edu/mishra04defect.html

[8] C. M. Jeffery, A. Basagalar, and R. J. O. Figueiredo, "Dynamic Sparing and Error Correction Techniques for Fault Tolerance in Nanoscale Memory Structures," in *4th IEEE Conference on Nanotechnology*, 2004.

[9] J. Han and P. Jonker, "A Fault-Tolerant Technique for Nanocomputers: NAND Multiplexing," *Journal of Nanotechnology*, vol. 1, December 2002. [Online]. Available: citeseer.ist.psu.edu/563079.html

[10] D. K. Bhavsar, "An Algorithm for Row-Column Self-Repair of RAMs and Its Implementation in the Alpha 21264," in *ITC '99: Proceedings of the 1999 IEEE International Test Conference*. Washington, DC, USA: IEEE Computer Society, 1999, p. 311.

[11] M. Nicolaidis, L. Anghel, and N. Achouri, "Memory Defect Tolerance Architectures for Nanotechnologies," *Journal of Electronic Testing: Theory and Applications*, vol. 21, no. 4, pp. 445–455, 2005.

[12] V. Schöber, S. Paul, and O. Picot, "Memory Built-In Self-Repair Using Redundant Words," in *ITC '01: Proceedings of the 2001 IEEE International Test Conference*. Washington, DC, USA: IEEE Computer Society, 2001, p. 995.

[13] E. Ou and W. Yang, "Fast Error-Correcting Circuits for Fault-Tolerant Memory," in *MTDT*, 2004, pp. 8–12.

[14] S. Lin and D. J. Costello, *Error Control Coding, Second Edition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004.

[15] J. Vollrath, U. Lederer, and T. Hladschik, "Compressed Bit Fail Maps for Memory Fail Pattern Classification," *J. Electron. Test.*, vol. 17, no. 3-4, pp. 291–297, 2001.

[16] M. Ziegler and M. Stan, "CMOS/nano Co-Design for Crossbar-Based Molecular Electronic Systems," *IEEE Transactions on Nanotechnology*, vol. 2, pp. 217– 230, 2003.

[17] M. B. Tahoori, "A mapping algorithm for defect-tolerance of reconfigurable nano-architectures," in *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pp. 668–672.

[18] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970. [Online]. Available: citeseer.ist.psu.edu/bloom70spacetime.html

[19] A. Naeemi and J. D. Meindl, "Design and Performance Modeling for Single-Walled Carbon Nanotubes as Local, Semiglobal, and Global Interconnects in Gigascale Integrated Systems," *IEEE Transactions on Electron Devices*, vol. 54, pp. 26–37, January 2007.

[20] S. Hamdioui, A. Goor, and M. Rodgers, "March SS: A Test for All Static Simple RAM Faults," in *IEEE International Workshop on Memory Technology, Design, and Testing*, 2002. [Online]. Available: citeseer.ist.psu.edu/hamdioui02march.html

[21] K. Zarrineh and S. J. Upadhyaya, "Programmable Memory BIST and a New Synthesis Framework," in *Symposium on Fault-Tolerant Computing*, 1999, pp. 352–355. [Online]. Available: citeseer.ist.psu.edu/254649.html

[22] D. Strukov and K. Likharev, "Defect-Tolerant Architectures for Nanoelectric Crossbar Memories," *Journal of Nanoscience and Nanotechnology*, p. 151167, January 2007.

[23] E. Witchel, J. Cates, and K. Asanović, "Mondrian Memory Protection," in *Proceedings of ASPLOS-X*, Oct 2002.

[24] Y. Massoud and A. Nieuwoudt, "Modeling and design challenges and solutions for carbon nanotube-based interconnect in future high performance integrated circuits," *Journal of Emerging Technologies in Computing Systems*, vol. 2, no. 3, pp. 155–196, 2006.