Efficient Storage of Defect Maps for Nanoscale Memory

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

Abstract

Nanoscale technology promises dramatic increases in device density, but reliability is decreased as a side-effect. With bit-error rates projected to be as high as 10%, designing a usable nanoscale memory system poses a significant challenge. Storing defect information corresponding to every bit in the nanoscale device using a reliable storage bit is prohibitively costly. Using a Bloom filter to store a defect map provides better compression at the cost of a small false positive rate (usable memory mapped as defective). Using a list-based technique for storing defect maps performs well for correlated errors, but poorly for randomly distributed defects. In this paper, we propose an algorithm for partitioning correlated defects from random ones. The motivation is to store the correlated defects using rectangular ranges in a ternary content-addressable memory (TCAM) and random defects using a Bloom filter. We believe that a combination of Bloom filter and small size TCAM is more effective for storing defect map at high error rate. We show the results for different correlated distributions.

1 Introduction

The microelectronic industry is facing difficult challenges related to extending integrated circuit technology beyond the scaling limit of CMOS[9]. Nanometer technology will have smaller, faster transistors, but greater sensitivity to defects such as copper voids, lattice dislocations, parasitic leakage etc[4]. Moreover, increases in cross-coupling capacitance and mutual inductance will have a severe effect on the yield of memory devices.

There have been two approaches in building nanoscale devices, namely self-assembly and lithography. In self-assembly, nanostructure materials such as carbon nanotubes (CNT) are assembled in defined locations with reproducible properties. Lithography refers to top-down building methodology where masks are used to fabricate devices. As device size continues to decrease and manufacturing costs continue to increase, the self-assembly is predicted to become more popular. Nanoscale devices, however, are expected to have high defect rates. These defects can be roughly divided into two classes: (i) permanent defects caused by inherent physical uncertainties in the manufacturing process, and (ii) transient faults due to lower noise tolerance or charge injection at reduced voltage and current levels. While the exact manufacturing defect rate is not yet known, defect rates as high as 10% have been reported[14]. This is more than eight orders of magnitude worse than the the rate found in current CMOS technology.

A number of defect-tolerant design methods have been proposed [16, 21, 7, 24] to deal with high error rates. These methods either use redundancy such as N-fold Modular Redundancy (NMR) [8] which is a simple form of Error Correcting Codes (ECC) [12], or reconfiguration in post manufacturing process to map out the defective regions. Error correction provides better reliability, but at very high error rates the probability of a component being defective also increases. Therefore, a defect map is required to ensure correctness.

For a nanoscale memory system, the overhead of keeping a reconfiguration bit for every non-reliable memory bit negates the density advantage offered by nanoscale memory devices as the overhead becomes 100%. Moreover, the reconfiguration bit has to be reliable (large) making the effective overhead much greater than 100%. Therefore, a more compact way of storing the reconfiguration data is needed. Wang et al. [24] proposed the use of a Bloom filter [2] for storing the defect map and evaluated their scheme for uniformly-random defects. For correlated defects, however, keeping a list of free regions may be more efficient.

In this paper, we focus on designing reliable memory systems using efficient defect maps. We focus on two mechanisms: a ternary content-addressable-memory (TCAM) for mapping regions of correlated errors and a Bloom filter for mapping uniformly random errors. We demonstrate that a combination of these two

mechanisms is a promising approach for addressing a wide range of defect scenarios in nanoscale technologies. We use the defect map on the level of a block of bits which reduces the overhead to a large extent. Error correcting codes (ECC) are used to provide reliability at the block level by reducing block defect rate.

The rest of the paper is organized as follows: In section 2, we give a description of the related research work. In Section 3, we give an overview of our system model and assumptions. Our methodology and results are given in Sections 4 and 5 respectively. Finally, we conclude with a perspective of our future work.

2 Related Work

In this section, we first describe the work done in the application of error-correction codes for developing reliable memory. Then, we provide an overview of the research in reconfiguration-based approaches. We describe techniques to store defect maps and argue for using our technique.

Error-correcting schemes have been widely used for both memory architectures and communication beginning with Von Neumann's seminal work on repetition codes [15]. State-of-the-art CMOS and disk technologies, however, have very small error rates which are in order of one in a billion and rigorous error correction is not always necessary. Jeffery et al.[10] proposed a 3-level error correcting memory architecture for nanoscale memory using single- or double-error correcting codes, 4^{th} level RAID and sparing of RAID arrays. For high error rates, however, stronger and multiple error correcting codes such as BCH codes are required as investigated by Sun et al. for nano-scale devices [21]. Ou et al. [17] proposes hardware design for the decoding and encoding routines of Hamming codes, where the memory reliability is increased at the cost of only $5ns$ delay in the memory access time. Hamming codes, however, are capable of correcting a single error in the block of physical bits used in the encoding, and they become less productive for high error rates. e.g. even by using BCH(250,32,45) code which provides 99.9956% correctness at 10% bit error rate in memory, 1 Byte in every 711 Bytes is expected to be defective. If we use only error correcting codes, we will require very strong and complex error correction codes resulting in large overhead in area and latency, and, therefore, we lose all the benefit of using nanoscale memory.

Therefore, in addition to active error correction through encoding, we need use defect maps to store the locations of the faulty bits in memory devices [23]. For reconfigurable architectures, tile-based memory units have been proposed where components store the defect map in a distributed fashion[8, 5, 25].The drawback of using defect maps in the bit-level is that the storage overhead is usually very high. Tahoori [22] proposes a *defect unaware design flow* which identifies universal defect free subsets within the partially defective chips, which reduces the size of the required defect map. Wang et al. [24] proposes the use of Bloom filters for storing defect maps for nanoscale devices. Hashing for every bit, however, is expensive computationally and may significantly increase the memory access times. The authors in [21] propose the use of employing CMOS memory for storing metadata to identify good parts 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 good part of the non-reliable memory and storing this index in the reliable CMOS. The amount of memory to store the ranges increases with the sparseness of faulty memory bits. It can be shown that when the error rate is close to 10%, the number of entries in the list is very large.

Error correcting codes reduce the defect rate of memory with the added cost of computation and redundancy. Strong error correcting codes (e.g. BCH(250,32,45)) are computationally expensive. The encoding and decoding delay is very high. We, therefore propose using less complex codes such as concatenation of Hamming and TMR which produces 90% correct blocks in presence of 10% bit error rate. In this paper, we propose a combination of Bloom filter and TCAM based defect map where correlated defects are captured by the TCAM entries and random defects are stored in Bloom filter.

3 Defect Model

Proposals to scale memory into the nanoscale regime have involved both non-silicon and silicon technologies. In non-silicon approaches, various techniques have been proposed among which most popular ones are carbon nanotubes(CNT), molecular switches, Spin Logic Devices, DNA etc. The key challenges with CNT based-devices are to:

1. Grow nanotubes (NTs) and nanowires (NWs) with predefined electronic properties through control of diameter, structure, and composition,

2. Position these structures in predefined locations and orientations, which may require sub-nanometer registration, and

3. Form contacts and interfaces with desired electronic properties and adhesion.

Recent development in CNT technique achieves 90% growth coverage.

Molecular electronic devices are built on the the idea of tailoring electronic properties of individual molecules to perform logic operations and on the assembly of a large number of these functional building blocks into molecular circuits. Two-terminal devices such as resistive switches as well as three-terminal devices such as gated, transistor-like molecules are envisioned.

Silicon based approaches focus primarily on hybrid technology e.g. CMOL. The basic idea of CMOL circuits is to combine the advantages of CMOS technology (including its flexibility and high fabrication yield) with the extremely high potential density of molecular-scale two-terminal nanodevices. Relatively large critical dimensions of CMOS components and the bottom-up approach to nanodevice fabrication may keep CMOL fabrication costs at affordable level. The error rate in CMOL devices have been found to be around 15%

CMOS scaling and manufacturing defects are somewhat better understood than non-silicon approaches. Due to its small size, nanoscale device are highly prone to process disturbance which results in manufacturing defects. Some of the disturbances are local which lead to randomly distributed small defects. Global defects include layer misintegration and line width variation. We model defects as a combination of uniformly random and correlated defects. Uniformly random defects are derived from the model of Maly[13]. However, faults in VLSI circuits tend to occur in a clustered fashion as same defect spans over multiple elements in the circuit[3]. Stapper [20] observed the size of defects distribution and found that the frequency of defects follow bell-shaped curve. Kim et al. [11] proposed using Poisson distribution to model the yield of CMOS devices in presence of defect. In this work, we use Gaussian distribution as our correlated model.

Preliminary evidence suggests that a combination of correlated and uncorrelated errors is also likely to apply to non-silicon technologies. As experimental evidence develops, we will refine our model, but we believe that a combination of uniformly random and correlated defects will apply to a wide range of technologies.

4 Methodology

Defect maps can be stored using different techniques e.g. bit-map, Bloom filter, list of ranges(linear) or 2-D ranges in the form of rectangle. When the error rate is not very high, a Bloom-filter-based method performs better than bit-vector based techniques. 2-D ranges generally perform better than 1-D ranges, as the correlation is spatial in nature.

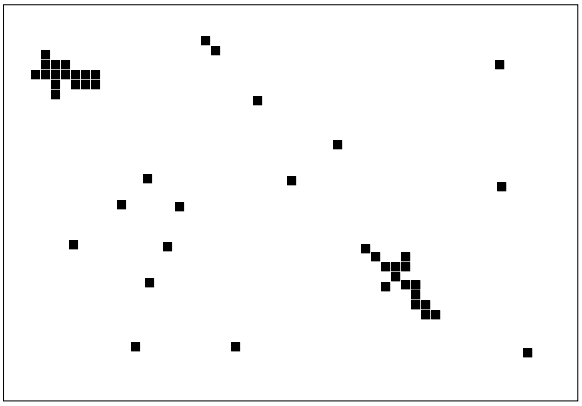


Figure 1: An example distribution of defects containing clustered and random defects

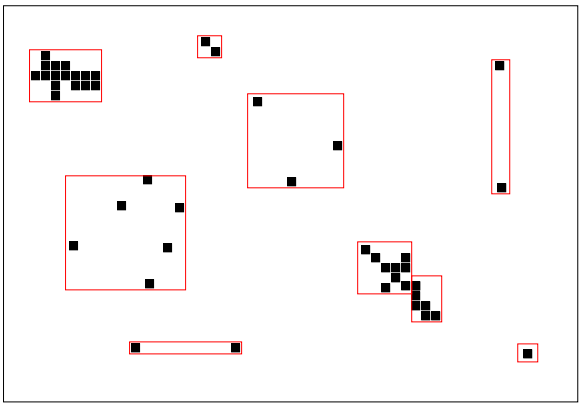


Figure 2: Storing defect map using 2-D ranges in TCAM. The red boxes indicate the regions.

We compare three schemes in this paper. In the first scheme, the defect map is stored in a TCAM. A Bloom filter is used in the second scheme and a combination of TCAM and Bloom filter is used in the third scheme. In the following subsections we explain each technique.

4.1 Scheme 1: TCAM Only

A TCAM provides the capability of fast searching in a parallel fashion, including range searching[19].

We use our TCAM entries to store rectangular ranges, and our goal is to find rectangular regions which cover a large number of defects without covering a large

Algorithm 1 R-Tree Based Rectangulation to obtain n rectangular regions covering all defects

1: Create $k - mean$ clusters with n means
2: Insert points in R-Tree starting with points closest to means
3: Obtain set of n rectangles R using Algorithm 2

amount of usable memory. In Figure 1, we show a sample distribution of defects having clustered and random defects. The algorithm 1 obtains n rectangles covering all the defective region. An R-Tree[6] data structure is modified to contain two children per parent node. The property of R-Tree that new data is added to that branch of the tree which needs minimal increase in rectangle size makes it suitable for obtaining the ranges from the defect map. The problem of finding the optimal cover of rectangles can be mapped to $0/1 - Knapsack$ problem which is $NP - Complete$ in nature. This algorithm is an approximate greedy solution which targets to cover all the points with a minimal false positive rate such as shown in Figure 2.

Algorithm 2 Choose n rectangular regions covering all defects

1: $R \leftarrow \phi$
2: $R \leftarrow root.branch[0].rectangle$
3: $R \leftarrow root.branch[1].rectangle$
4: **while** number of rectangles $\leq n$ **do**
5: Replace the branch with its children, which reduces the false positive rate at maximum
6: **end while**

4.2 Scheme 2: Bloom filter Only

A Bloom filter [2] is a hash-based data structure that offers a compact way to store a set of items to support membership queries. A Bloom filter works as follows: A set of n elements $S = \{s_1,\ s_2, \ldots,\ s_n\}$ is mapped to the Bloom filter vector B of m bits by a set of k independent hash functions, $\{H_1,\ H_2, \ldots,\ H_k\}$. Each item in the set S is hashed k times, with each hash yielding a bit location in the Bloom filter string that is set to 1. To check if element x belongs to the set, we can hash it k times and check if all of the corresponding bits in the Bloom filter are set to 1, otherwise x is not in the set.

The space efficiency of Bloom filters comes with some percentage of false-positives since x may hash to bits in the Bloom filter that have been set by a different element. Therefore, Bloom filters are good data structures when membership queries are needed from a stored list of items, memory is important, and the effect of some false-positives can be mitigated. The false-positive rate is a function of the length of the Bloom filter string as it relates to the number of the hashed keys (n), the number of hash functions (k), and the hash functions' ability to evenly populate the Bloom filter.

Compared to keeping a bit-vector for storing the defect map, Bloom filter performs better for low and moderate error rate. If the block error rate is r, number of blocks is N, the memory overheard for keeping 1 bit per b bit block is $\frac{N}{b}$ blocks. The memory overhead for Bloom filter is frN bits i.e. frN/b blocks where $f = m/n$ for bloom filter. For false positive rate of 5.6%, $f = m/n = 6$. Therefore, the overhead of Bloom filter is less when the Block error rate is $\leq 1/6$ i.e. 17%. The sparser the Bloom filter string, the better the false-positive rate. In the paper submitted to iccad07[1], we described a modification in hashing technique which performs close to the theoretical limit using reduced computational resource. At $m/n = 20$ the false-positive rate drops to approximately $0.18\% \pm 0.06\%$ percent and still take less than 6.5% of the total Memory. Perhaps most importantly, using this simple hash function, we can implement a fast Bloom filter suitable for memory architectures.

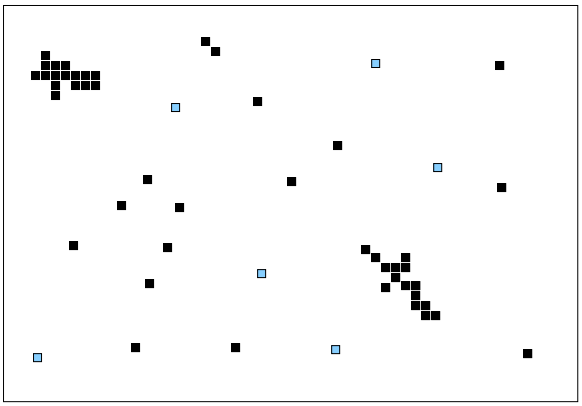


Figure 3: Storing defect map using Bloom filter. Blue boxes are false positives.

In Figure 3, we show the effect of using Bloom filter for storing the defect map. The light-shaded boxes indicate the blocks which are marked as defective though they are functional. The false positive rate increases when the correlation in defect decreases as shown in result section, but it performs equally to a small TCAM. Therefore, we propose to use the combination of TCAM and Bloom filter as described in the following subsection.

4.3 Scheme 3: TCAM and Bloom filter

In this scheme we identify the correlated defects from random defects by pruning the R-Tree obtained

in algorithm 1. We choose one node from the list of nodes obtained as result of algorithm 1 to remove from the list of node and replace with one of its children. This node is chosen greedily such that $\frac{decrease\ in\ false\ positive}{decrease\ in\ number\ of\ points\ from\ the\ set}$ is minimum as described in Algorithm 3. The scheme is illustrated by Figure 4 where the red boxes indicate TCAM regions and other defects are covered by Bloom filter. We show the performance in section 5.

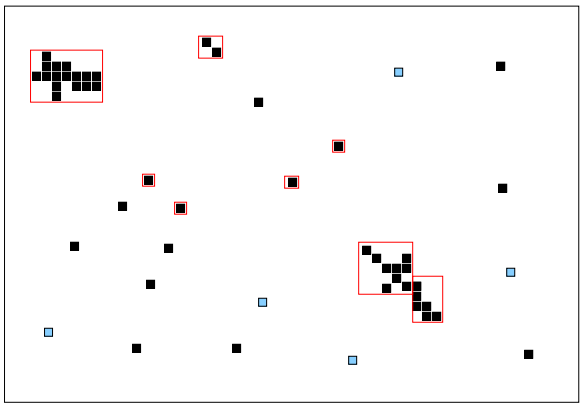


Figure 4: Storing defect map in a combination of Bloom filter and 2-D ranges. Clusters are stored in 2-D ranges and Bloom filter stores sparse points.

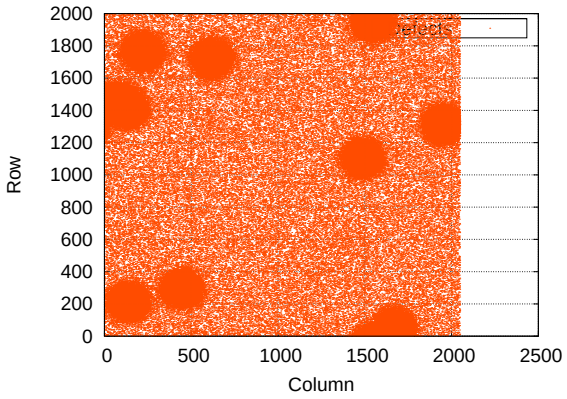
Algorithm 3 Choose n rectangular regions which covers maximal points with false positive below a threshold

1: **repeat**
2: Set target threshold for TCAM
3: Run algorithm 1
4: **while** false positive rate $\geq$ threshold **do**
5: Replace the branch with one its children, which has maximum $\frac{decrease\ in\ false\ positive}{decrease\ in\ number\ of\ points\ from\ the\ set}$
6: Insert points from other branch in Bloom filter
7: **end while**
8: Compute overall false positive
9: **until** false positive $\leq$ global threshold

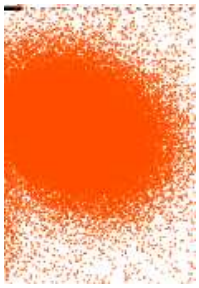
5 Results

Using error correcting codes(ECC) at the block level (e.g. 32-bits) reduces the effect of random defects as illustrated in Figure . Therefore, clustered defects become more significant.

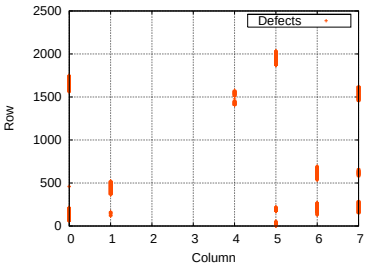
We tested our algorithms with a 128-entry TCAM on a 2M-Byte nanomemory with 10% bit error. For 100%



(a) Bit level distribution



(b) Zoomed in view of a cluster



(c) Block level distribution

Figure 5: Distribution of Defects as a combination of correlated and random defects

correlated defects, a smaller TCAM could be used, but for random errors its performance degrades quickly. If the ranges are stored in a backing store, then the TCAM could serve the purpose of a cache and such an organization will be the subject of future work. The simulation of the Bloom filter was computationally intensive, so our prelimary study focuses on a 2M-Byte memory. The Bloom filter was chosen to be 5 times the number of defects, i.e. $m/n = 5$. Effectively, the size of the Bloom filter is 1.56% of the whole memory. As mentioned in Section 3, we used a combination of random and clustered error model. The clustered defects were correlated by Gaussian distribution. We tested the performance of the schemes we proposed for different proportion of clustered and random errors as well as with different degree of correlation. Figure 6 shows the results with varying proportion of correlated and random defects. We can observe that the combined scheme offers high accuracy when there is significant spatial correlation in defects.

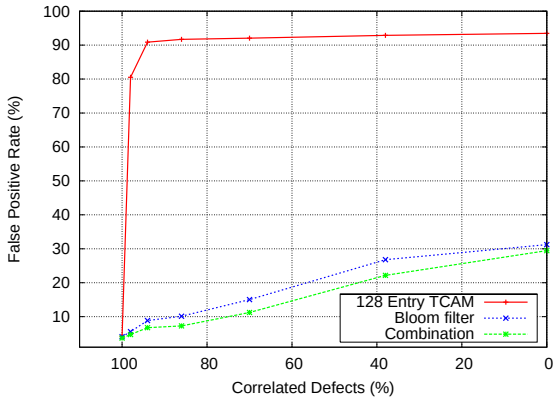


Figure 6: Comparison of false positive rate of three schemes with random defects. False positive rate of TCAM-based approach increases very fast with randomness as the number of entries are small.

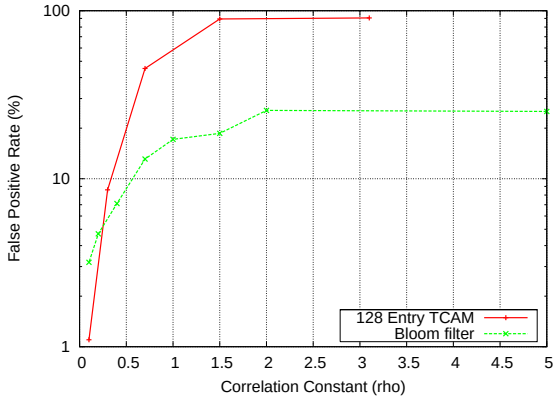


Figure 7: Comparison of false positive rate of TCAM and Bloom filter based schemes with correlation constant of defect distribution. False positive rate of TCAM-based approach increases very fast with randomness as the number of entries are small.

Figure 7 illustrates that the TCAM-based approach is effective only at high correlation. As we can observe from Figure 6, however, the 128-entry TCAM is comparable in performance to Bloom filter whose size is 1.56% of the memory to achieve 10% false positive. The usefulness of the TCAM decreases with decreasing correlation in defect distribution as illustrated in Figure 8.

6 Conclusion

In this paper, we propose to use a combination of a Bloom filter and a small TCAM to store the defect map of a nanoscale memory in a compact way. Error-correcting codes employed at the block level reduce the effect of random defects to great extent, leaving a combination of clustered and random defects. We proposed a greedy algorithm to partition the points in TCAM and Bloom filter. We obtained the population in the rectangular regions as shown in Figure 9. Our current algorithm generates large rectangles with low population as it tries to use all the rectangles. Hence, the result can be more enhanced if all the rectangles are not used. We will focus on this area in our future work.

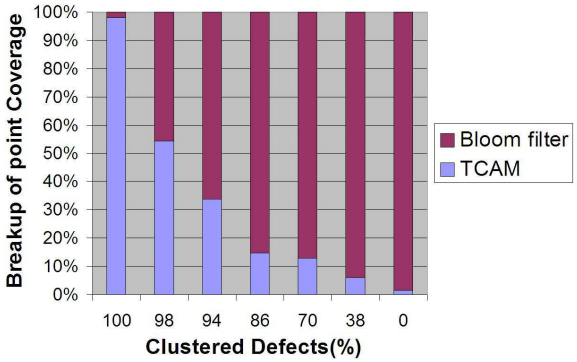


Figure 8: Defects covered by TCAM and Bloom filter shown with variation in randomness of defects. Contribution of TCAM is low when randomness of defect is high. Bloom filter performs better in that stage

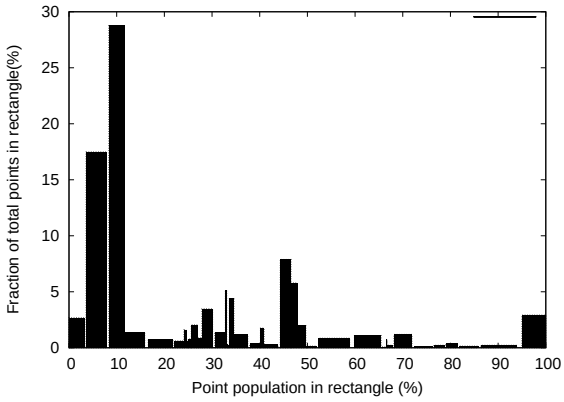


Figure 9: Distribution of points in rectangles. More populated the rectangle, better is its utilization

As TCAM cells are significantly more complex than RAM cells, the number of TCAM cells cannot be very large. Selective-precharge-based techniques can be used to reduce the power requirement of TCAM[18], but to compete with the Bloom filter map, the TCAM area must be small. We have shown that a small TCAM can efficiently store the defect map when the defects are clustered. Using larger number of regions, the accuracy of 1^{st} approach can be improved. Future work will examine a hierarchical scheme in which ranges are stored in a backing store and cached in a small TCAM.

7 Future Work

We observe that Bloom filter which we designed performs better for correlated data. Universal hash functions are used in Bloom filter which ensures uniform performance for all kinds of data distribution. In ideal case, we want to have the Bloom filter performing better for random defects than correlated defects. Moreover, we observe that we need large size Bloom filter map when the memory size is large. We plan to cache the Bloom filter results in order to improve performance. To enhance the performance of the 1^{st} approach, we need to have larger list where TCAM can serve as cache for storing the regions. Due to locality of memory references small TCAM might be able to perform satisfactorily.

References

[1] S. Biswas, T. S. Metodi, G. Wang, F. T. Chong, and R. Kastner. Combining static and dynamic defect-tolerance techniques for nanoscale memory systems. *Submitted to ICCAD 2007, Available at http://cs.ucsb.edu/∼susmit/papers/iccad-nano07.pdf*.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] D. Blough and A. Pelc. A clustered failure model for the memory array reconfiguration problem. *IEEE Transactions on Computers*, 42(5):518–528, 1993.

[4] L. Chen, S. Dey, P. Sanchez, K. Sekar, and Y. Chen. Embedded hardware and software self-testing methodologies for processor cores. In *Proceeding of the 37th Design Automation Conference*, pages 625 – 630, June 2000.

[5] 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*, pages 375–382, 2005.

[6] A. Guttman. R-trees: a dynamic index structure for spatial searching. pages 599–609, 1988.

[7] J. Han, J. Gao, Y. Qi, P. Jonker, and J. A. B. Fortes. Toward hardware-redundant, fault-tolerant logic for nanoelectronics. *IEEE Des. Test*, 22(4):328–339, 2005.

[8] 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.

[9] ITRS. *International Technology Roadmap For Semiconductors - 2006 Edition*. Semiconductor Industry Association, 2006.

[10] 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.

[11] T. Kim and W. Kuo. Modeling manufacturing yield and reliability. *Semiconductor Manufacturing, IEEE Transactions on*, 12(4):485–492, 1999.

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

[13] W. Maly. Realistic fault modeling for vlsi testing. In *DAC '87: Proceedings of the 24th ACM/IEEE conference on Design automation*, pages 173–180, New York, NY, USA, 1987. ACM Press.

[14] M. Mishra and S. Goldstein. Defect tolerance at the end of the roadmap. In *ITC*, pages 1201–1211, 2003.

[15] J. V. Neuman. Probabilistic logic and the synthesis of reliable organisms from unreliable components. *Automata Series, Editors: C. Shannon and J. McCarthy, Princeton Univ. Press*, pages 43–98, 1956.

[16] G. Norman, D. Parker, M. Kwiatkowska, and S. K. Shukla. Evaluating the reliability of defect-tolerant architectures for nanotechnology with probabilistic model checking. In *VLSID '04: Proceedings of the 17th International Conference on VLSI Design*, page 907.

[17] E. Ou and W. Yang. Fast error-correcting circuits for fault-tolerant memory. In *MTDT*, pages 8–12, 2004.

[18] V. Ravikumar, R. N. Mahapatra, and L. N. Bhuyan. Easecam: An energy and storage efficient tcam-based router architecture for ip lookup. *IEEE Transactions on Computers*, 54(5):521–533, 2005.

[19] S. Sharma and R. Panigrahy. Sorting and searching using ternary cams. *HOTI*, 00:101, 2002.

[20] C. H. Stapper. On yield, fault distributions, and clustering of particles. *IBM J. Res. Dev.*, 30(3):326–338, 1986.

[21] 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.

[22] 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*, pages 668–672.

[23] J. Vollrath, U. Lederer, and T. Hladschik. Compressed bit fail maps for memory fail pattern classification. *J. Electron. Test.*, 17(3-4):291–297, 2001.

[24] G. Wang, W. Gong, and R. Kastner. Defect-tolerant nanocomputing using bloom filters. In *ICCAD 2006*, November 2006.

[25] M. Ziegler and M. Stan. CMOS/nano co-design for crossbar-based molecular electronic systems. *IEEE Transactions on Nanotechnology*, 2:217–230, 2003.