time. Divide memory modules into two groups A and B. Place x_{2k} in group A and x_{2k-1} in group B, $k = 1, 2, \cdots$. The computation of linked list prefix requires the communication of $n/2$ data items between groups A and B while the interface between two groups has only $o(p)$ cells. Therefore, timing $O(n/p)$ cannot be achieved.

Another way to weaken the local memory PRAM model is to limit the degree of each processor. The degree of a processor is the number of processors it is directly connected to. The degree of a parallel computer is the maximum of the degrees of its processors. A p-processor local memory PRAM with p shared cells corresponds to a parallel machine of p processors with an interconnection network connecting every pair of processors. The degree of each processor in this machine is p. However, reducing the degree of a computer could significantly weaken its communication power [6]. There are n input data items and n output data items, a permutation from the input data items to the output data items is required for computing the linked list prefix problem. The results of Gottlieb and Kruskal [6] show that with a computer of degree k $\Omega\left(\frac{n \log_k p}{p}\right)$ is a lower bound for static permutation. When the computer has degree $k = 2^{o(\log p)}$, timing $O(n/p)$ cannot be achieved.

Finally, the memory sharing scheme of the p memory cells could possibly be weakened for computing the linked list prefix. We are not clear if it is possible to achieve time $O(n/p + \log n)$ without concurrent access to the p shared cells.

Acknowledgment

I wish to thank anonymous referees for pointing out an error in the original manuscript.

References

[1] R. J. Anderson and G. L. Miller, "Optimal parallel algorithms for the list ranking problem," U.S.C. Tech Rep. 1986.
[2] ——, "Deterministic parallel list ranking," in *Lecture Notes in Computer Science 319, VLSI Algorithms and Architectures,* J. Reif, Ed., 3rd Aegean Workshop on Computing, 81-90, June–July, 1988.
[3] R. A. Borodin and J. E. Hopcroft, "Routing, merging and sorting on parallel models of computation," in *Proc. 14th ACM Symp. Theory Comput.,* San Francisco, CA, Apr. 1982, pp. 338–344.
[4] R. Cole and U. Vishkin, "Deterministic coin tossing and accelerating cascades: Micro and macro techniques for designing parallel algorithms," in *Proc. 18th ACM Symp. Theory Comput.,* 1986, pp. 206–219.
[5] ——, "Approximate and exact parallel scheduling with applications to list, tree and graph problems," in *Proc. 27th Symp. Foundations Comput. Sci.,* IEEE, 1986. pp. 478–491.
[6] A. Gottlieb, C. P. Kruskal, "Complexity results for permuting data and other computations on parallel processors," *J. ACM,* vol. 31, no. 2, pp. 193–209, Apr. 1984.
[7] Y. Han, "Designing fast and efficient parallel algorithms," Ph.D. dissertation. Dep. Computer Sci., Duke Univ., 1987.
[8] ——, "Parallel algorithms for computing linked list prefix," *J. Parallel Distributed Comput.,* vol. 6, pp. 537–557, 1989.
[9] C. P. Kruskal, T. Madej, and L. Rudolph, "Parallel prefix on fully connected direct connection maching," in *Proc. 1986 Int. Conf. Parallel Processing,* pp. 278–284.
[10] C. P. Kruskal, L. Rudolph, and M. Snir, "The power of parallel prefix," *IEEE Trans. Comput.,* vol. C-34, no. 10, pp. 965–968, Oct. 1985.
[11] T. Leighton, "Tight bounds on the complexity of parallel sorting," *IEEE Trans. Comput.,* vol. C-34, 344–354, 1985.
[12] G. L. Miller, and J. H. Reif, "Parallel tree contraction and its application," in *Proc. 26th Symp. Foundations Comput. Sci.,* IEEE, 1985, pp. 478–489.
[13] G. F. Pfister and V. A. Norton, "Hot spot contention and combining in multistage interconnection networks," *IEEE Trans. Comput.,* vol. C-34, no. 10, pp. 934–948, Oct. 1985.
[14] J. H. Reif, "An optimal parallel algorithm for integer sorting, in *Proc. 26th Symp. Foundations Comput. Sci.,* IEEE, 1985 pp. 291–298.
[15] ——, "Probabilistic parallel prefix computation," in *Proc. 1984 Int. Conf. Parallel Processing,* Aug. 1984.
[16] M. Snir, "On parallel searching," *SIAM J. Comput.,* vol. 14, no. 3, pp. 688–708, Aug. 1985.
[17] R. A. Wagner and Y. Han, "Parallel algorithms for bucket sorting and the data dependent prefix problem," in *Proc. 1986 Int. Conf. Parallel Processing,* pp. 924–930.
[18] J. C. Wyllie, "The complexity of parallel computation," TR 79-387, Dep. Comput. Sci., Cornell Univ., Ithaca, NY, 1979.

A RAM Architecture for Concurrent Access and On-Chip Testing

Jyh-Charn Liu and Kang G. Shin

***Abstract*— A new RAM architecture supporting concurrent memory access and on-chip testing (CMAT) is proposed. A large-capacity memory chip is decomposed into *test neighborhoods* (TND's), each of which is tested independently of the others. When there are data stored in a TND, the data are saved into a buffer before testing the TND, and the TND's contents are restored using buffered data after testing the TND. If an external request is not made to the TND, the request can be directed to the addressed memory cells. Otherwise, the buffered data can be loaded back into the TND, or the request is detoured to a corresponding buffer. By deriving an analytical model, the performance penalty and hardware overhead of the CMAT architecture are shown to be very small.**

***Index Terms*— Address mapping mechanism (AMM), built-in tester (BIT), concurrent memory access and on-chip testing (CMAT), mean fault detection time, memory access locality, single-cell pattern sensitive fault (SPSF), single decoder fault, test neighborhood (TND).**

I. Introduction

For critical applications such as avionic control and life support systems, a crash in the computer system could lead to an economic disaster and/or loss of human lives. Thus, any computer system designed for critical applications must have a high degree of fault tolerance. A hardware *fault* is defined as an incorrect state caused by the physical change in a component, whereas an *error* is defined to be the incorrect information/data resulting from the manifestation of a fault. The period from the occurrence of an error (fault) to its detection is called the *error (fault) detection time.* While it is relatively easy to diagnose and recover from a single fault, it is extremely difficult and costly to handle multiple faults [1].

For real-time applications, a shorter fault detection time means a longer time available to take appropriate actions (e.g., damage assessment and recovery) against the fault before the associated deadlines, if any, expire. To shorten the mean fault detection time with negligible performance penalties, we propose a new architecture which supports

Manuscript received June 4, 1987; revised September 12, 1990. This work was supported in part by NASA under Grants NAG-1-296 and NAG-1-492. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of NASA.

J.-C. Liu is with the Department of Computer Science, Texas A&M University, College Station, TX 77843.

K. G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 9143283.

0018–9340/91$01.00 © 1991 IEEE

concurrent memory access and on-chip testing (CMAT) of permanent faults. Thus, the proposed memory architecture will henceforth be called the CMAT architecture. In the CMAT architecture, the contents of memory cells are saved before testing them, and a memory chip can be tested concurrently, but not simultaneously, with normal memory access. A memory matrix is decomposed into *test neighborhoods* (TND's), and each TND is tested independently. When a TND is to be tested, its contents are first saved in a buffer, and then test patterns are applied to the TND. After the TND is tested, its buffered data are loaded back into the corresponding memory cells.

Most related work only addresses off-line testing of memory chips. Single-cell pattern-sensitive fault (SPSF) model is most widely accepted memory fault model for its practicality [2]–[4]. Single error correcting and double error detecting (SECDED) codes for main memory have been widely used in large computer systems. It has been reported that if coding methods were used as the sole means of error detection/correction, then a very long (one hour or more) fault detection time in main memory may result [5]. The long fault latency with SECDED codes can be reduced by memory scrubbing [6], [7]. However, unless more error correction bits are added to the system, performance penalties of memory scrubbing increase with memory size. Other related work includes built-in testers (BIT's) for testing different memories [8]–[10], and parity encoding [11].

The remainder of this correspondence is divided into four sections. Section II presents design details for testing different parts of memory. The performance of the CMAT architecture is analyzed in Section III. Finally, concluding remarks are made in Section IV.

II. CMAT Architecture

Memory components are tested in two parts: peripheral circuits and memory cells. The testing task is assumed to be free-running, and a BIT tests a memory matrix or chip autonomously.

A. Architectural Characteristics

When the capacity of a RAM chip is very large, the chip is usually made of several memory matrices, where each of them is an $N \times N$ bit matrix. Let C_{ij} denote the memory cell located at the (i, j) position of a memory matrix. Each matrix has the same structure, say, 8K bits of memory cells, a row decoder (RD), a column decoder (CD) and sense amplifiers, as shown in Fig. 1. A CMAT architecture consists of a BIT, peripheral circuits, and memory cells. The row decoder, column decoder, and sense amplifiers are collectively called the peripheral circuit of a memory matrix. The BIT is composed of an address mapping mechanism (AMM), a buffer, an ASND mirror (to be defined later), a column neighborhood (CN) and a row neighborhood (RN), and comparators for verifying the outputs of TND's.

In the CMAT architecture, data may be stored in the regular memory cells or in the buffer(s). Since both external access requests and the BIT may read/write data, data consistency between the regular memory cells and the buffer(s) must be maintained in the CMAT architecture. There are two strategies to achieve data consistency. The first is that, when the TND is requested, the testing procedure is suspended, and all the data stored in the buffer are loaded back into the TND. After the data are loaded back into the TND, normal access operations in the memory are resumed. The second strategy is that, when the TND is requested, the request is detoured to the buffer. For the purpose of exploring potential design complexity, only the more complicated case of detouring request to the buffer will be considered here.

An AMM is used to map a TND address to a buffer address, and vice versa. When an external memory access request is not made to the TND being tested, the request can be serviced as if it were made to regular cells. When the TND is requested, the current memory access cycle is either extended so that data in the buffer can be loaded back into the regular cells, or the request is detoured to the buffer. Thus, the system must allow for variable memory access times.

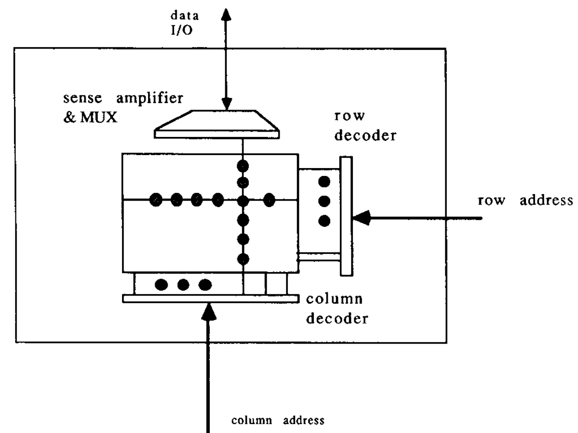

Fig. 1. A two-and-half-dimensional RAM organization.

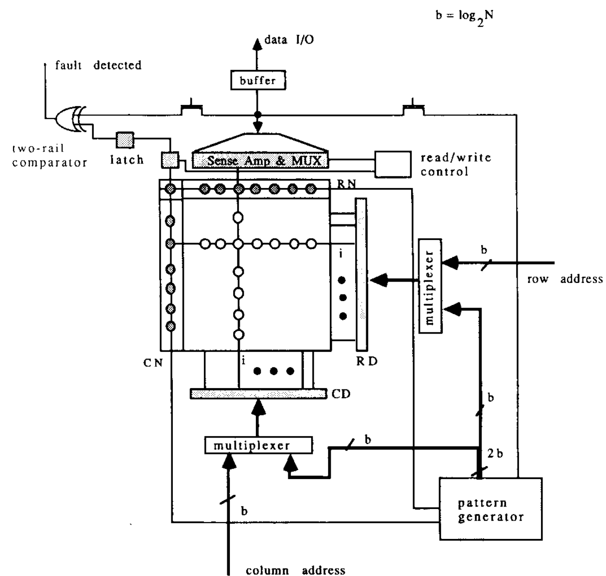

Fig. 2. Design for peripheral circuit testing.

B. Peripheral Circuit Testing

The peripheral circuits include a row decoder, a column decoder, and sense amplifiers. A single decoder fault occurs when one of the row or column decoders, but not both, is faulty. A row (column) neighborhood, denoted by RN (CN), is a collection of cells in a row (column). To test single decoder faults, dedicated row and column neighborhoods are proposed. As shown in Fig. 2, the row (column) neighborhood is made independent of the original row (column) decoder by using an independent row (column) enable line. The row (column) decoder is tested by reading from, and writing to, the column (row) neighborhood.

The testing procedure first sets row and column neighborhoods to known initial states, and then various test patterns, such as those in [2] and [12], can be readily applied. Note that test patterns cannot be applied to row and column decoders simultaneously. Thus, when one test pattern is applied to the row (column) decoder, all column (row) decoder outputs are disabled and the column (row) neighborhood is enabled by an independent signal. Each read/write operation generated from the pattern generator is applied, say, to the row decoder first, and then to the column decoder. For a read operation, the ith cell in the column neighborhood is first read and stored in a latch, and then the ith cell in the row neighborhood is read and compared to the value in the latch. For a write operation, the ith cells in the row and column neighborhoods can be written in two consecutive cycles. When the testing pattern proposed in [2] is used for column and row testing, the length of the testing procedure for decoders is $2|TD|$, where $|TD| = N(2N^2+3N+4)$ memory cycles.

A slow recovery sense amplifier is detected by a long stream of 1's (0's) followed by a 0 (1): each cell in the column neighborhood is connected to a distinct sense amplifier. The ith sense amplifier can be tested by the test pattern $\overline{W}_i^k W_i R_i W_i^k \overline{W}_i R_i$, where W_i $(\overline{W}_i)$ is to write 1 (0) to an arbitrary memory cell connected to amplifier i, R_i is to read the corresponding memory cell, and k is the sense amplifier's time constant. To test a sense amplifier, the row decoder is disabled first, and then one of the column decoder outputs, the CN-enable, and the RN-enable lines are enabled simultaneously. As shown in Fig. 2, the test pattern is applied simultaneously to the ith sense amplifier $(0 \leq i \leq N-1)$ and the column neighborhood's sense amplifier for the purpose of demonstration. The outputs of the two sense amplifiers can then be verified by a two-rail comparator. It will take $N(2k+4)$ cycles to test sense amplifiers. Thus, it needs a total of $N(2N^2+3N+2k)$ memory cycles to test the peripheral circuits.

C. Memory Cell Testing

For memory cell testing, an *augmented single-cell test neighborhood* (ASND) is used in this correspondence. An ASND is a 3 × 3 matrix composed of nine memory cells. As will be seen later, use of ASND's can greatly simplify the BIT design. SPSF's are to be tested in the memory cells, and testing outcomes are verified by comparing the outputs of two identical circuits. Thus, a set of test patterns are applied to an ASND and an *ASND mirror* simultaneously, where the ASND mirror has exactly the same number of memory cells as that of an ASND. The testing length for the memory matrix is N^2K, where K is the length of a selected test pattern for an ASND.

Since data patterns are stored in the memory cells, data must be loaded/saved properly during testing of these memory cells. An ASND with the center located at C_{ij} is denoted by ASND_{ij}, $\text{ASND}_{ij} = \{C_{km} | i-1 \leq k \leq i+1, j-1 \leq m \leq j+1\}$. ASND_{ij} can be decomposed into three rows or columns, i.e., $\text{ASND}_{ij} = \bigcup_{k=1}^{3} SR_{ij}^k$ or $\text{ASND}_{ij} = \bigcup_{k=1}^{3} SC_{ij}^k$, where SR_{ij}^k (SC_{ij}^k) is the kth row (column) of ASND_{ij}.

Starting with ASND_{11}, the data in $\{\text{ASND}_{ik} | i \text{ is odd}, 0 \leq k \leq N-1\}$ are saved, and then loaded from left to right. For simplicity, boundary ASND's are not included in the testing procedure. After an odd numbered row i is tested, SR_{iN-2}^1 is loaded into memory and SR_{i+1N-1}^3 is saved into the buffer. Every even numbered row is loaded from right to left, and SR_{i1}^1 (SR_{i+10}^3) is loaded (saved) when a row is tested. This sequence is described by the following algorithm, pseudo-coded in C.

Save ASND_{11} into buffer;
for $(i=1; i \leq N-2; i++)\{$
$D = (-1)^{i+1}$;
$L = \frac{N-3}{2} + \frac{N-3}{2}D + 1$;
$U = \frac{N-3}{2} - \frac{N-3}{2}D + 1$;
$x = 2 + D$;
$y = 2 - D$;
for $(j = L; j \neq U; j = -D)\{$
test ASND_{ij};
load SC_{ij}^x;
save SC_{ij+D}^y;}
load SR_{ij}^1;
save SR_{i+1j}^3;}.

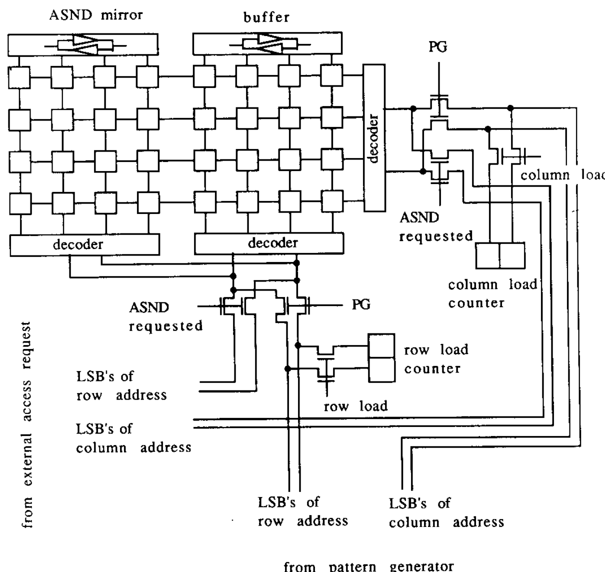

Fig. 3. Design of the buffer, ASND mirror, and AMM for the serial testing of memory cells.

It takes six memory cycles to load and save data between two adjacent ASND's. It needs $6(N^2-4N+4)$ memory cycles for the above load–save procedure.

According to the load/save procedure, the AMM and the buffer must perform correct address mapping without physically moving the data in the buffer. A design which can simplify the AMM's structure is shown in Fig. 3. Both the buffer and the ASND mirror are expanded into a 4 × 4 matrix, and the least significant two bits of the address bus can be used directly to access the buffer and/or the mirror. Since each data load operation moves a row or column, i.e., SR^i or SC^i, of data into the regular memory cells, two up–down counters are sufficient to indicate the row and column to be loaded.

The memory organization for the testing of SPSF's is given in Fig. 4. The total length of testing procedure, t_D, including the data load/save procedure, is $t_D = 2N(2N^2+3N+4) + N(2k+4) + N^2K + 6(N^2-4N+4)$ memory cycles. For example, when $k=10$, and the test pattern in [2] is used, $K = 2720$, we get $t_D = 4N^3 + 2732N^2 + 8N + 24$ memory cycles. Additional circuits necessary to implement CMAT are the row and column neighborhoods, pattern generators, the multiplexer between the external address and the test pattern from the pattern generator, etc; $2N + 4|\text{ASND}|$ extra memory cells are needed to implement CMAT. The hardware overhead of memory cells is $\frac{2N+64}{N^2}$, which is less than 0.9% for $N = 256$, and 2% for $N = 128$. It should be noted that use of multiplexers is essential to any built-in testing architecture. Thus, the hardware overhead will be dominated by pattern generators.

Fig. 4. Serial testing for SPSF's in a RAM matrix.

Fig. 5. Design of the buffer and AMM for the parallel testing of memory cells.

D. Parallel Testing of Memory Cells

To further speed up memory testing, the sense amplifiers can be modified to allow parallel testing of memory cells.

The parallel testing design in [8] can be easily modified for the CMAT architecture. The buffer of a CMAT chip for testing SPSF's is composed of four rows. The shift control gates are used for both loading/saving buffers in the CMAT architecture and shifting test patterns for the testing procedure. However, before testing a row of $\{\text{ASND}_{ij} | 1 \leq j \leq N\}$, the shift control gates are used to load its data into the buffer. After testing $\{\text{ASND}_{ij} | 1 \leq j \leq N\}$, the contents of the nonoverlapping row are restored with the buffered data. At the next test cycle, similar load–test–save procedures are repeated. A block diagram for parallel testing in the CMAT architecture is shown in Fig. 5. When data are loaded into the buffer, their physical locations are left-skewed by one column position. Thus, when an external request needs data in the buffer, the data must be loaded back into the original memory cells first.

It takes N cycles to both load and save data in the parallel testing method. The length of parallel testing procedure for peripheral circuits is the same as its serial testing counterpart. However, the testing speed of memory cells is N times faster than that of serial testing. Thus, $t_D = 2|TD| + N(2k+4) + 2N + NK$. When $K = 2720$ and $k = 10$, we get $t_D = 4N^3 + 6N^2 + 2754N$. The parallel testing method needs four columns of memory cells for the buffer, shift control gates, an AMM, a pattern generator, and a two-rail comparator. While the overhead of most components is determined by the design implementation, the overhead for buffer is $\frac{4}{N}$.

III. Performance Analysis

Based on a new memory access model, the computational loss and fault detection time of CMAT are computed in this section. However, the multiplexing delay is excluded from our discussion, because such delay is inevitable for all built-in testing methods.

A. Memory Access Model

The existence of memory access locality has long been recognized in computer systems. That is, once a program starts to access a specific memory area, it tends to access the area continuously for a certain period. A program's behavior of memory access can be modeled by an *active agent* visiting several chips and forming access set(s). An *access set*, AS_i^j, is defined as a memory area that is continuously accessed for a certain period at the active agent's ith visit to chip j. For tractability reasons, it is assumed that 1) all memory chips have the same structure and testing strategy, 2) an access set does not cross chips and its lifetimes are independent and identically distributed (i.i.d.), and 3) the agent's present and future visits to chips are independent of its past visits.

There are m chips in the system, and the system is said to be in *state* i when the agent is in chip i. Let V_k^i represent the event of the agent's kth visit to chip i, and M_k^i be the lifetime of V_k^i. Let S_k^i denote the time V_k^i begins, and $N^i(t) = \sup\{k | S_k^i \leq t\}$. Since M_k^i's

are assumed to be i.i.d., $\{N^k(t), t \geq 0\}$ is a renewal process, and $\{N^1(t), N^2(t), \cdots, N^m(t), t \geq 0\}$ is a Markovian renewal process. Let $Z(t)$ denote the system state at time t, i.e., the agent is in chip $Z(t)$ at time t. Then, $\{Z(t), t \geq 0\}$ is a semi-Markov process. A random variable $J(i) = j$ is used to represent that the agent's ith visit is made to chip j. By Assumption 3), $\{J(i), i \geq 1\}$ is a Markov chain. For a given time interval $[0, t), t > 0$, the total number of the agent's visits to chips is $N(t) \equiv \sum_{k=1}^{m} N^k(t)$.

There may exist n different distributions for M_k^i's in each chip. Let D_j^i denote the agent's selection of a memory access time distribution at its jth visit to chip i and let $D_j^i = k$ when the distribution F_k is selected, where $P(D_j^i = k) = \alpha_k$. The normal memory access time during the agent's jth visit to chip i is denoted by X_j^i. Thus, $F_{X_j^i | D_j^i = k}(t) = F_k(t)$.

When the agent is in chip i, it may or may not access the TND being tested. An indicator function I_j^i associated with V_j^i represents whether or not the agent accesses the TND; $I_j^i = 1$ (0) when the agent accesses (does not access) the TND. The probabilities of $I_j^i = 1$ and $I_j^i = 0$ are denoted by $P(I_j^i = 1) = \beta$ and $P(I_j^i = 0) = 1 - \beta, \forall i, j$, respectively. β is determined by $\frac{|\text{ASND}|}{N^2}$, where |ASND| is the number of memory cells in an ASND and N^2 is the size of the memory matrix. $M_j^i = X_j^i$ when the agent does not access the TND being tested, and $M_j^i = X_j^i + Y_j^i$ when the agent accesses the TND, where Y_j^i is a random variable representing the delay of request-detouring or data-load/save. Thus, the distribution function of M_j^i is $F_{M_j^i}(t) = (1 - \beta) \sum_{k=1}^{n} \alpha_k F_k(t) + \beta \sum_{k=1}^{n} \alpha_k \int_0^t f_{X_j^i + Y_j^i | D_j^i = k}(\xi)\, d\xi$, where f_x is the density functions of a random variable X. For the tractability of analysis, we shall consider only the case when X_j^i and Y_j^i are independent.

B. Computational Loss

The computational loss due to the load/save procedures is determined by the system workload and testing strategies used. From the agent's point of view, it makes no difference to which chip the performance penalty occurs.

Since both Y_j^i's and M_j^i's are assumed to be i.i.d., $E(Y_j^i) = E(Y)$, and $f_{M_j^i} = f_M, \forall i, j$, where Y is a representative random variable for Y_j^i's and M for M_j^i's. Now, we want to evaluate the ratio of access delay to the normal memory access time as a relative performance penalty. For this, the system model is refined with two states A and W. The system is in a) state A when the agent accesses chip i without hitting the TND being tested, or b) state W when the agent accesses the TND and, thus, waits until the data are loaded/saved. The system can then be modeled as a regenerative process; the system regenerates itself as it enters state W then $A, A, \cdots$, then W then $A \cdots$. The time interval between two consecutive regenerative instants is called a *cycle*, denoted by C, whose length is T_C. Denote the time interval for the system staying in states A and W by random variables T_A, and T_W, respectively, the expected computational loss then becomes $P_W = E(T_W)/E(T_C)$, where $E(T_W)$ is the average time that the system is in state W. Since $E(T_C) = E(T_W) + \beta \sum_{i=1}^{\infty} i(1 - \beta)^{i-1} E(T_A) = E(T_W) + E(T_A)/\beta$, we get

$$P_W = 1 \Big/ \left\{1 + \frac{E(T_A)}{\beta E(T_W)}\right\}. \tag{3.1}$$

Assume that the width of both row and column addresses is b, i.e., $N = 2^b$, $\beta = \frac{5}{2^{2b}}$, and $\beta = \frac{3}{2^b}$, for the serial and parallel testing of memory cells, respectively. Performance loss of the parallel testing method is plotted in Fig. 6. Note that for the same parameters as that of parallel testing, the performance penalty of CMAT architecture is negligible ($\ll 2\%$) even under an extremely conservative condition that $E(T_A) = E(T_W)$. Since the load/save procedure of the parallel testing needs only two memory access cycles, $E(T_A)/E(T_W)$ is expected to be very high. Thus, the performance penalty of the parallel testing can also be kept very low.

C. Fault Detection Time

The basic assumptions for the analysis of fault detection time in the CMAT architecture are summarized as follows: 1) the length of testing procedure, t_D, is fixed, 2) a memory chip is tested continuously by its BIT, and 3) normal memory access is given priority over the testing operation. Given the occurrence of a fault, the *testing length*, $t_L \leq t_D$, is defined as the time to detect the fault with the BIT in a noninterruptive manner. When a fault occurs in chip i, the fault detection time is actually $T_L^i = t_L + E^i$, where E^i is a random variable representing the total normal memory access time of chip i before the fault is detected.

To derive the distribution of T_L^i, one must know exactly when and where a fault occurs. It is assumed that a fault may occur at any time with an equal probability, and the effects of the BIT's testing on memory access times are assumed to be negligible in the analysis of fault detection time, i.e., $M_i^j \approx X_i^j, \forall i, j$. Since $\{Z(t)\}$ is a semi-Markov process, the steady-state probability of the agent being in chip i is $P_i = \prod_i \mu_i / \sum_{k=1}^{m} \prod_k \mu_k$, where $\prod_i = \lim_{j \to \infty} P(J(j) = i)$, and $\mu_i = \lim_{n \to \infty} \sum_{j=1}^{n} M_j^i / n$, the agent's mean sojourn time in chip i.

Viewing the system from the BIT in chip i, events $V_1^i, \overline{V}_1^i, V_2^i, \overline{V}_2^i \cdots$ occur sequentially, where $\overline{V}_j^i = \bigcup_{n,k} V_k^n$, $n \neq i$, and $S_j^i < S_k^n < S_{j+1}^i$. Let $\overline{M}_j^i$ denote the lifetime of $\overline{V}_j^i$, we can get $f_{R_k^i}(y) = \frac{1 - F_{M_k^i}(y)}{E(M_k^i)}$ [13], where $R_k^i(t)$ is the remaining lifetime of V_k^i at time t.

A fault may occur during V_k^i or $\overline{V}_k^i$. When a fault in chip i occurs during V_k^i, the fault cannot be detected before the active agent leaves the chip. After the agent leaves the chip, the chip may be visited n times, $0 \leq n \leq \infty$, before the fault is detected. Thus, the event requiring a testing length t_L to detect the fault is $\bigcup_{n=1}^{\infty} \{\sum_{j=k}^{k+n} \overline{M}_j^i \geq t_L, \sum_{j=k}^{k+n-1} \overline{M}_j^i < t_L\} \bigcup \{\overline{M}_k^i \geq t_L\}$. Thus, $T_L^i = R_k^i + t_L$ for $\overline{M}_k^i \geq t_L$, and $T_L^i = R_k^i + \sum_{j=k+1}^{k+n} M_j^i + t_L$ for $n \geq 1$. Derivation of an exact form for $F_{T_L^i}$ is in general intractable. However, for a given $n \geq 0$, the upper bound of T_L^i is $t_L + \sum_{j=k}^{k+n} M_j^i$ by moving the point of fault occurrence to S_k^i.

We now consider the case when a fault occurs during $\overline{V}_k^i$. The fault may be detected during $\overline{V}_k^i$, or the chip may be visited by the active agent n times, $1 \leq n \leq \infty$, before the fault is detected. Thus, for a given t_L, the event of fault detection is

$$\left\{\overline{R}_k^i \geq t_L\right\} \bigcup \left\{\overline{R}_k^i + \overline{M}_{k+1}^i \geq t_L, \overline{R}_k^i < t_L\right\} \bigcup_{n=2}^{\infty} \cdot \left\{\overline{R}_k^i + \sum_{j=k+1}^{k+n} \overline{M}_j^i \geq t_L, \overline{R}_k^i + \sum_{j=k+1}^{k+n-1} \overline{M}_j^i < t_L\right\}$$

where $T_L^i = t_L$ if $\overline{R}_k^i \geq t_L$, and $T_L^i = \sum_{j=k+1}^{k+n} M_j^i + t_L$ if $n \geq 1$.

It is easy to determine the distribution of T_L^i if $\overline{M}_j^i$'s are governed by a Poisson process. When $f_{\overline{M}_k^i} = \mu e^{-\mu t} \quad \forall\, k, i$ we get $f_{\overline{R}_k^i} = f_{\overline{M}_k^i}$. When a fault in chip i occurs during V_k^i, for a given t_L the probability of detecting the fault during $\overline{V}_{k+n}^i$ is $P_{k+n}(t_L) = \frac{e^{-\mu t} L(\mu t_L)^n}{n!}$, $n \geq 0$. Since M_j^i's are assumed to be i.i.d., $E(M_j^i) = E(M^i) \quad \forall\, j$. The expected fault detection time for a given t_L is

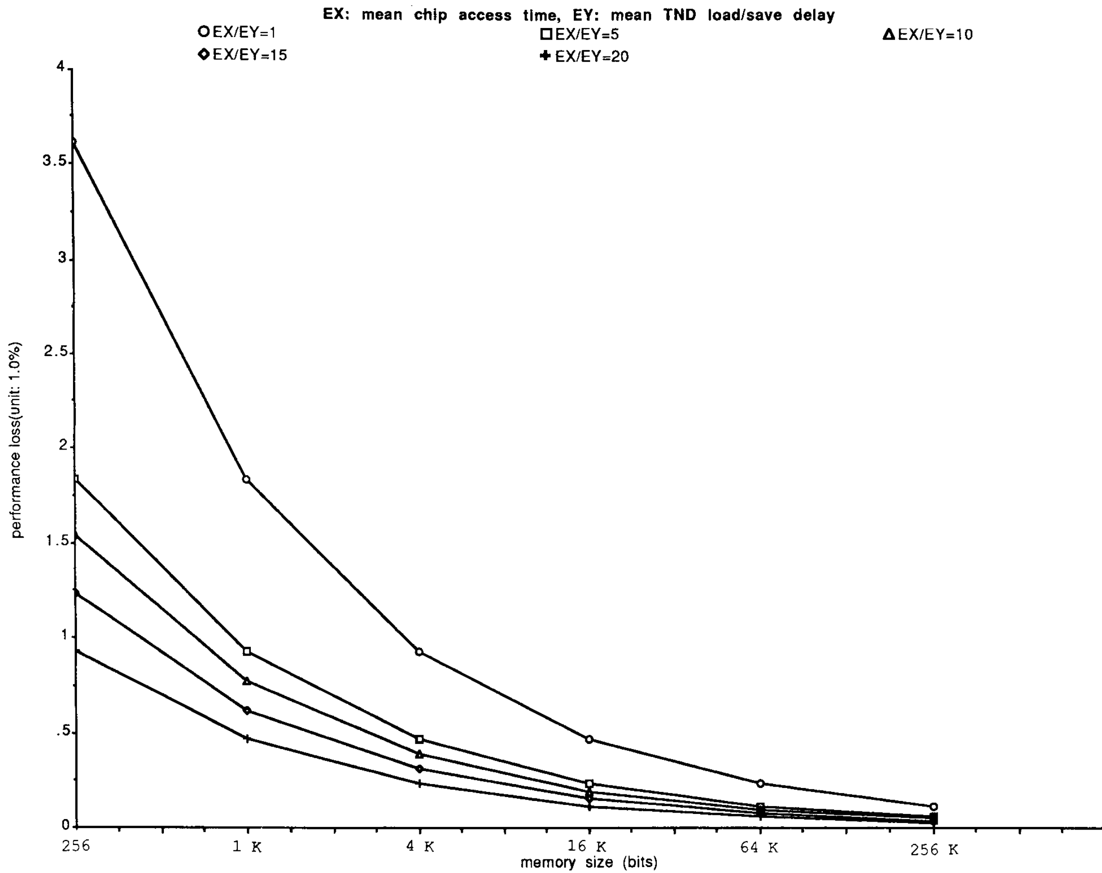

Fig. 6. Performance loss of the CMAT architecture with a parallel testing scheme.

then

$$E\left(T_L^i|t_L\right) = E\left(R_k^i\right) + t_L + \sum_{n=1}^{\infty} P_{k+n}(t_L) n E\left(M^i\right)$$
$$= E\left(R_k^i\right) + \left(1 + \mu E\left(M^i\right)\right) t_L.$$

Similarly, when a fault occurs during $\overline{V}_k^i$, the probability of detecting the fault during $\overline{V}_{k+n}^i$ is $P_{k+n}(t_L) = \frac{e^{-\mu t_L}(\mu t_L)^n}{n!}$, $n \geq 0$, because $f_{\overline{M}_k^i} = f_{\overline{R}_k^i}$, and thus the distribution of fault detection time is $F_{T_L^i}(t_L + t) = \sum_{n=0}^{\infty} P_{k+n}(t_L) \int_0^t f_{M^i}^{n-1}(\xi)\, d\xi$, where $f_{M^i}^j$ is the j-fold self-convolution of f_{M^i}. Thus, the conditional mean fault detection time is $E(T_L^i|t_L) = [1 + \mu E(M^i)]t_L$. Using the assumption that the distribution of t_L is uniform, we get

$$E\left(T_L^i\right) = P_i \left[E\left(R_k^i\right) + \frac{1}{t_D}\int_0^{t_D}\left\{1 + E\left(M^i\right)\mu\right\} t\, dt\right] + (1 - P_i)\left[\frac{1}{t_D}\int_0^{t_D}\left\{1 + E\left(M^i\right)\mu\right\} t\, dt\right]$$
$$= P_i E\left(R_k^i\right) + \frac{t_D}{2}\left(1 + \mu E\left(M^i\right)\right). \qquad (3.2)$$

$E(R_k^i)$ can be expressed in terms of $E(M^i)$ as $E(R_k^i) = \frac{E((M^i)^2)}{2E(M^i)}$. The length of test pattern for parallel testing is N times shorter than that of serial testing. Thus, when $E(R^i)$ is small, and $E(M^i) \ll \frac{1}{\mu}$, the mean fault detection time of parallel testing scheme is nearly N times shorter than that of serial testing scheme.

Through similar computational steps as that of the CMAT architecture, performance of SECDED codes with scrubbing can also be analyzed. A computer system with $M_s = 16$ megabytes of main memory, $|AS| = 1$ kilobytes, and $t_a = 1$ minute is used as an example. For typical parameter values $\lambda = 10^{-5}$, 32-bit data width and 7-bit error correction codes, mean fault detection time for the coding methods without scrubbing is approximately 4.7×10^6 h. When the whole memory is scrubbed once every T_d time units, each time taking $T_s \leq T_d$, the mean fault detection time is $T_d/2$, and the performance loss is T_s/T_d.

Consider a case where each matrix contains 64 Kbits, the memory cycle is 200 ns, and the example testing procedure in Section III-B is used. Then, the mean fault detection time of the CMAT architecture is approximately 50 s, with a negligible performance penalty. The mean fault detection time can be reduced to 200 ms when a parallel testing scheme is used in the CMAT architecture. The performance loss of scrubbing can become substantial when a short mean fault detection time is important. For example, when the test patterns in [7] is used, the performance loss would be 27%, as shown in Fig. 7, for a coding method with scrubbing to obtain the same mean fault detection time

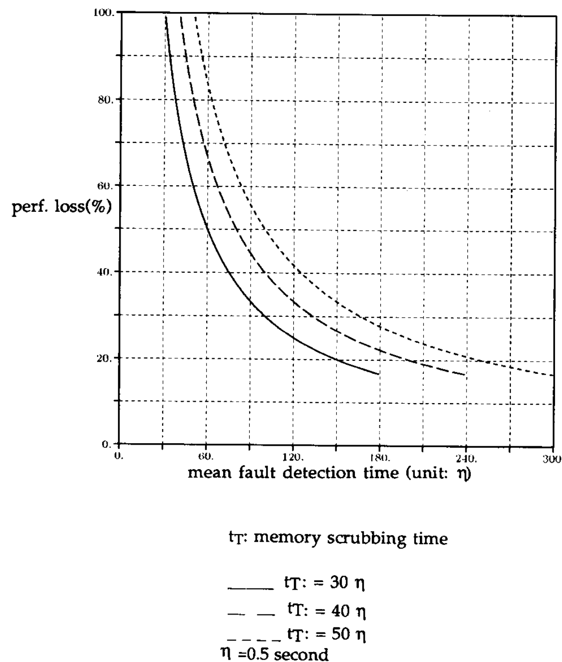

Fig. 7. The performance loss of SECDED codes with scrubbing versus mean fault detection time.

as that of the CMAT testing scheme. SECDED codes along with scrubbing would be sufficient if the arrival rate of failures is low and the cost of system crash is not too high. However, if both latent faults and performance loss are serious concerns, the conventional ECC plus scrubbing technique will quickly reach its limit, and other techniques like CMAT or adding more error correction/detection bits to the system should be considered.

IV. Conclusion

Although early fault detection may not have direct impacts on system reliability, it will make the subsequent fault handling procedures (e.g., fault containment and recovery) much easier. The CMAT architecture is intended to ease the problem of testing large-capacity memory chips with little hardware overhead and performance loss. The distribution function of fault detection time appears to be very complex. In order not to distract readers from our main intent, i.e., the CMAT architecture, derivation of a closed-form expression for general distributions was not pursued. An optimal testing strategy for each application needs to be derived to meet its specific requirements. The expected fault detection time can be controlled by adjusting the length of testing procedure, the length that the agent stays in a chip, and the interarrival time of the agent at each chip. An optimal testing strategy should also consider the different fault coverage and hardware overhead of different designs. The analysis of system behavior becomes much more complicated in that case.

References

[1] K. G. Shin and Y.-H. Lee, "Evaluation of error recovery blocks used for cooperating processes," *IEEE Trans. Software Eng.*, pp. 692–700, 1984.
[2] J. P. Hayes, "Testing memories for single-cell pattern-sensitive faults," *IEEE Trans. Comput.*, vol. C-29, pp. 249–254, Mar. 1980.
[3] K. K. Saluja and K. Kinoshita, "Test pattern generation for api faults in RAM," *IEEE Trans. Comput.*, vol. C-34, pp. 284–287, Mar. 1985.
[4] D. S. Suk and S. M. Reddy, "Testing procedures for a class of pattern-sensitive faults in semiconductor random access memories," *IEEE Trans. Comput.*, pp. 419–429, June 1980.
[5] R. Chillarege and R. K. Iyer, "Fault-latency in the memory—An experiment on vax 11/780," in *Dig. Papers, FTCS-16*, 1986, pp. 258–263.
[6] J. F. J. Aichelmann, "Fault-tolerant design techniques for semiconductor memory applications," *IBM J. Res. Develop.*, vol. 28, no. 2, pp. 177–183, Mar. 1984.
[7] D. Rennels and S. Chau, "A self-exercising self-checking memory design," in *Dig. Papers, FTCS-16*, 1986, pp. 358–363.
[8] Y. You and J. P. Hayes, "A self-testing dynamic ram chip," *IEEE J. Solid-State Circuits*, vol. SC-20, no. 1, pp. 428–435, Feb. 1985.
[9] K. Kinoshita and K. K. Saluja, "Built-in testing of memory using an old-chip compact testing scheme," *IEEE Trans. Comput.*, vol. C-35, pp. 862–870, Oct. 1986.
[10] Z. Sun and L.-T. Wang, "Self-testing of embedded rams," in *Proc. Int. Test Conf.*, 1984, pp. 148–156.
[11] R. M. Tanner, "Fault-tolerant 256k memory designs," *IEEE Trans. Comput.*, vol. C-33, pp. 314–322, Apr. 1984.
[12] J. P. Hayes, "Detection of pattern-sensitive faults in random-access memories," *IEEE Trans. Comput.*, vol. C-24, pp. 150–157, Feb. 1975.
[13] L. Kleinrock, *Queueing Systems Vol. 1: Theory.* New York: Wiley, 1975.

Branch Strategies: Modeling and Optimization

Pradeep K. Dubey and Michael J. Flynn

***Abstract*— Instruction dependency introduced by conditional *branch* instructions, which is resolved only at run-time, can have a severe performance impact on pipelined machines. A variety of strategies are in wide use to minimize this impact. Additional instruction traffic generated by these branch strategies can also have an adverse effect on the system performance. Therefore, in addition to the likely reduction a branch prediction strategy offers in average branch delay, resulting excess *i*-traffic can be an important parameter in evaluating its overall effectiveness. The objective of this paper is twofold: to develop a model for different approaches to the branch problem and to help select an optimal strategy after taking into account the additional *i*-traffic generated by branch strategies.**

The model presented provides a flexible tool for comparing different branch strategies in terms of the reduction it offers in average branch delay and also in terms of the associated cost of wasted instruction fetches. This additional criterion turns out to be a valuable consideration in choosing between two almost equally performing strategies. More importantly, it provides a better insight into the expected overall system performance. Simple compiler-support-based low implementation-cost strategies can be very effective under certain conditions. An active branch prediction scheme based on loop buffer can be as competitive as a branch-target-buffer based strategy

***Index Terms*—Branch prediction, conditional branches, degree of dependency, instruction dependency, instruction traffic, pipelining.**

Manuscript received December 28, 1987; revised October 5, 1990. M. J. Flynn was supported by NASA under Contract NAG2-248.

P. K. Dubey is with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

M. J. Flynn is with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305.

IEEE Log Number 9143286.

0018–9340/91$01.00 © 1991 IEEE