Accurate Reliability Evaluation and Enhancement via Probabilistic Transfer Matrices

Smita Krishnaswamy †, George F. Viamontes ‡, Igor L. Markov †, John P. Hayes †
{smita,gviamont,imarkov,jhayes}@eecs.umich.edu
† Advanced Comp. Architecture Lab, University of Michigan, Ann Arbor 48109
‡ Theoretical Division, T-13, Los Alamos National Lab, NM 87545

Abstract

Soft errors are an increasingly serious problem for logic circuits. To estimate the effects of soft errors on such circuits, we develop a general computational framework based on probabilistic transfer matrices (PTMs). In particular, we apply them to evaluate circuit reliability in the presence of soft errors, which involves combining the PTMs of gates to form an overall circuit PTM. Information such as output probabilities, the overall probability of error, and signal observability can then be extracted from the circuit PTM. We employ algebraic decision diagrams (ADDs) to improve the efficiency of PTM operations. A particularly challenging technical problem, solved in our work, is to simultaneously extend tensor products and matrix multiplication in terms of ADDs to non-square matrices. Our PTM-based method enables accurate evaluation of reliability for moderately large circuits and can be extended by circuit partitioning. To demonstrate the power of the PTM approach, we apply it to several problems in fault-tolerant design and reliability improvement.

1 Introduction

Transient (probabilistic) errors have been a major source of system crashes for years. Traditional sources of transient errors include power supply instability and mismatches between components. As transistor sizes decrease, it becomes important to consider transient errors in logic circuits caused by cosmic rays [10]. When a primary particle strikes the atmosphere, neutrons and other secondary particles can be emitted. Neutrons can strike a critical node in a circuit, leaving behind an ionized track in silicon possibly causing a bit flip error to be latched. This necessitates methods to model and analyze circuits with probabilistic errors.

We model circuits at the logic level using a matrix representation for gates, an idea that goes back to [7]. We use the form given in [9] known as the *probabilistic transfer matrix* (PTM) formulation which represents parallel composition of gates with tensor products. Matrix representations of gates are illustrated in Figure 1, where the probability of each output value is explicit for each input combination.

Given individual gate error probabilities, we can compute output probabilities for the entire circuit and the overall probability of correctness, i.e., reliability of the circuit. This requires calculating the PTM for the whole circuit by combining gate PTMs. The process of combining gate PTMs implicitly takes into account signal dependencies between gates by considering the underlying joint and conditional probabilities within the circuit. Once the circuit PTM is calculated, accurate information about output probabilities, reliability and signal observabilities can be extracted from it.

Specific applications of interest include comparing the reliability of various circuits for a Boolean function and choosing critical components for partial duplication [8]. The

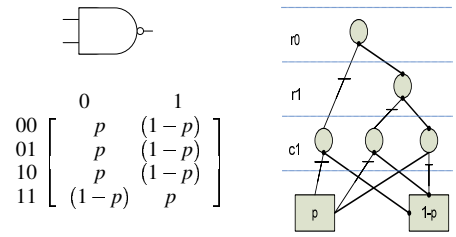

Figure 1: PTM (left) and ADD (right) representations of a NAND gate that produces an incorrect output with probability p for any given input. The labels $r0$, $r1$, and $c1$ in the ADD diagram represent row and column index variables.

exactness of the calculations can be used to make small, incremental enhancements to a circuit. While approximate methods exist, higher precision may be required to evaluate sensitivities to changes in the circuit structure. Exact techniques are also needed to automate the evaluation of fault-tolerant constructs such as NAND-multiplexing and cascaded triple modular redundancy (TMR) [13].

The PTM evaluation method has several advantages over existing methods of soft error modeling. It performs simultaneous computation over all possible input combinations. It calculates the exact probabilities of errors, so no input vector sampling is involved. Also we do not have to explicitly map out signal dependencies in the network beyond specifying the gate PTMs. Larger circuits can be analyzed by applying this technique to sub-circuits in conjunction with event-driven simulations or path tracing. Having subcircuits precomputed as PTMs can speed up these algorithms.

A potential memory bottleneck in the PTM approach is alleviated through the use of ADDs (algebraic decision diagrams) [3] to compress PTMs (see Figure 1). The ADD is reduced in that identical nodes are not represented twice. Unnecessary decisions are always skipped, and the variable ordering interleaves row and column variables. For a gate with n inputs and m outputs, the straightforward representation of its PTM requires space $O(2^{n+m})$. For a computer with 2GB memory, this limits the size of the circuit to 16 input/output signals in practice. However, the use of the ADD compression method significantly decreases the memory requirements in practice. PTM operations including information extraction are performed on the compressed form. Therefore, this method has complexity polynomial in the size of the ADD of a circuit and empirically scales to a width of 40+ signals. This facilitates exact analysis of circuit blocks used in regular circuit fabrics such as PLAs, FPGAs, memories and structured ASICs.

A major technical challenge in using ADDs for PTM representation is the lack of ADD algorithms for non-square matrices. Existing algorithms use zero-padding to make a matrix square. However, as we show, zero-padding gener-

ally produces incorrect results for a key operation needed in PTM evaluation. We circumvent this difficulty by using permutation matrices and fanin gates.

The remainder of this paper is organized as follows. Section 2 outlines related work. Section 3 introduces the PTM framework and operations required to calculate circuit PTMs. Section 4 explains how ADDs are used to compress PTMs, outlines our PTM evaluation algorithm and presents empirical results. Section 5 examines several applications, while Section 6 discusses conclusions and future work.

2 Previous Work

Prior research on probabilistic analysis of logic circuits can be broadly classified into these areas: probabilistic behavior modeling, circuit simulation, fault-tolerant architectures and reliability improvement by adding redundancy. Work has been done in modeling dependencies between signals in the form of a Bayesian network. The relation between circuit signals and Markov random fields (undirected Bayesian networks) is described by [2] in the context of probabilistic computation using neural networks. The conditional probability of outputs given input signals determines how errors are propagated through a circuit. Using this type of theoretical model it is possible to predict the probability of output error given the gate errors. However empirical results in [2] are not presented for any significant circuits .

The work in [1] describes a model for simulating the physical effects of soft errors on VLSI circuits. It uses an event-driven simulator with timing information to inject faults into individual gates. A transient error is latched depending on factors such as electrical, latching-window and logical masking. The functionality and connectivity of gates can make errors unobservable or influence the output error in various ways. The simulator injects faults for each test vector and determines whether the fault is latched despite the three basic sources of errors.

In [8] the soft error or single event upset (SEU) susceptibility of gates is estimated. The *soft error susceptibility* of a gate g with respect to a latch l, is calculated by $p_{error}(g,l) = R_{SEU} * P_{sense}(g,l) * P_{latched}(g,l)$. Here R_{SEU} is the probability that an incident neutron can produce an SEU. $P_{sense}(g,l)$ is the probability that an input sensitizes a particular gate. $P_{latched}(g,l)$ is the probability that the error occurs during the latching window of the clock cycle. In order to assign an error susceptibility to an individual gate, a set of input vectors (selected from sampling runs of a circuit) are tested along with a fault at the particular gate. Each fault is propagated through critical path tracing to assess its affect on the output. This gives the approximate probability that an error appearing at a particular gate is latched.

The work in [2] models probabilistic behavior of logic circuits by Bayesian networks, while our work relies on simpler modelling with matrices [7, 9]. Our data structures enable further scalability than was shown in [2].

Developments in nanotechnology have led to recent interest in fault-tolerant architectures [6]. In order to shrink devices reliably, techniques such as NAND multiplexing have been proposed. This technique involves analyzing the fault-tolerance of NAND multiplexing with Markov chains [6]. The analysis indicates that a low degree of redundancy may be sufficient for unreliable nanodevices. In our work, we wish to automate the analysis of fault-tolerant architectures using PTM evaluation. We do not require most of the assumptions used in analytical models and therefore can consider a wider range of probabilistic error patterns.

3 PTM Theory

We utilize the PTM model proposed in [9] which uses tensor products as well as matrix multiplication for notational convenience. In [7, 9] there was little discussion of implementation or applications of matrix based models of gate behavior. In this paper, we develop compuatational support for the PTM model and demonstrate practical results in applications.

The probabilistic behavior of a circuit can be described by a matrix A where the (j,k)th entry represents the probability of output signals $O = o_0, o_1, \ldots o_n$ having value k given that input signals $I = i_0, i_1 \ldots i_m$ have value j. This is denoted $p(O = k|I = j)$. Here the row and column indices j and k are thought of as bit vectors whose entries represent the values of the signals that form the input and output. For convenience we omit the signal names and write this as $p(k|j)$. For instance $p(1,1|1,0)$ represents the probability that the two output variables (O_0, O_1) have value $(1,1)$ given that the two input variables (I_o, I_1) have value $(1,0)$. Probability $p(1,1|1,0)$ can be alternately written as $p(3|2)$.

Definition 1 *A matrix M where the (j,k)th entry represents the probability of output value k given input value j, i.e., $p(k|j)$, is called a* probabilistic transfer matrix (PTM). *A fault-free circuit has an* ideal transfer matrix (ITM)*, i.e., the correct value of the output occurs with probability* 1.

Circuit PTMs are calculated from gate PTMs by combining gate PTMs in a manner dictated by their connectivity. We now describe the matrix operations needed to construct a circuit PTM from the PTMs of its components.

For two gates g_1 and g_2 with PTMs M_1 and M_2 that are connected in series, the entry $p(k|j)$ in the combined matrix represents the probability that g_2 produces output k in case g_1 is given input j. Therefore

$$p(k|j) = \sum_{all\ l} p(k|l)p(l|j)$$

This operation corresponds to the matrix product of the two component PTMs denoted M_1M_2.

Now consider two gates g_1 and g_2 with PTMs M_1 and M_2 that are connected in parallel. We view indices k and j of the entry $p(k|j)$ in the combined matrix as concatenated in binary from indices k_1, k_2 and j_1, j_2 of the two original PTMs. The value of $p(k|j)$ is the joint probability that the input j_1 of gate g_1 causes value k_1 at the output and, at the same time, the input j_2 of gate g_2 causes k_2 at the output of g_2. Therefore $p(k|j) = p(k_1,k_2|j_1,j_2) = p(k_1|j_1)p(k_2|j_2)$. This operation is commonly known as the *tensor product*.

Definition 2 *Given two matrices M_1 and M_2 with dimensions $m \times n$ and $o \times p$ respectively, the* tensor product *of M_1 and M_2, denoted $M_1 \otimes M_2$ is an $mo \times np$ matrix whose entries are given by $p(k|j) = p(k_1|j_1)p(k_2|j_2)$.*

We represent fanout behavior by explicit *fanout gates*. A n-output fanout gate, denoted F_n, copies an input signal to its n outputs. The ITM of a 2-output fanout gate is therefore given by $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

Wire permutations (where necessary) are represented by permutation matrices. Wire permutations are a special class of permutations which we call *variable permutations*. Variable permutation matrices are induced by permuting bits of row indices and column indices (these bits are known as row variables and column variables) of the identity matrix.

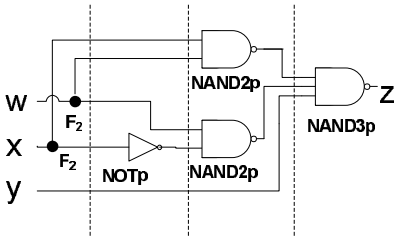

Figure 2: Circuit to illustrate the calculation of a PTM from gate PTMs. The junction dots represent fanout gates. The dotted lines separate levels of the circuit.

Definition 3 *For a matrix A consider a permutation* $\sigma(\ )$ *on the row variables. A row of A with index* $i_0 i_1 \ldots i_n$ *in binary is mapped to the row with index* $\sigma(i_0)\sigma(i_1)\ldots\sigma(i_n)$ *in binary. This defines a* row variable permutation *of A with respect to* σ*, which we denote as* $rperm(A,\sigma)$.

Since a wire permutation is a row variable permutation on the identity matrix it is denoted $rperm(I,\sigma)$. For instance the matrix for an adjacent wire swap is given by:

$$\begin{bmatrix} 1&0&0&0\\ 0&0&1&0\\ 0&1&0&0\\ 0&0&0&1 \end{bmatrix}$$

To calculate the PTM of an acyclic combinational circuit with known gate PTMs, we combine PTMs of smaller components into larger PTMs using matrix products and tensor products and wire permutations. Additionally, explicit fanout gates and identity gates (buffers) may be required to reconcile dimensions before applying matrix products. This bottom-up evaluation procedure is illustrated next.

Example 1 For the circuit shown in Figure 2, the 2-input NAND gates have the PTM $NAND2_p$ given in Figure 1, and the PTM $NAND3_p$ for the 3-input NAND gate is similar. An inverter with uniform error probability p has the PTM $NOT_p = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}$. The circuit PTM is expressed by:

$$(F_2 \otimes F_2 \otimes I)(I \otimes I \otimes NOT_p \otimes I)(rperm(I,(23)))(NAND2_p \otimes NAND2_p \otimes I)(NAND3_p) \quad (1)$$

where I refers to a 2×2 identity PTM and the wire permutation fixes wires $0,4$ and swaps wires $2,3$. Each parenthesized term in Equation 1 corresponds to a circuit level in Figure 2.

While other models for representing circuit errors exist, we use the PTM framework because of its generality and versatility. The PTM model can represent a wide variety of gate behavior including stuck-at faults and transient errors. The fact that there are separate probabilities for each input and output and the fact that they are propagated simultaneously enable this generality. Also the PTM model can be easily extended in various ways.

4 PTM Representation and Evaluation

The size of a gate PTM with n inputs and m outputs is $2^n \times 2^m$. The memory needed to store PTMs can be reduced by compressing them and operating on the compressed forms. This section discusses compression by encoding the matrix into an algebraic decision diagram (ADD) and develops an efficient procedure for evaluating circuit PTMs.

4.1 Compressing Matrices with ADDs

In general, all entries of a $2^n \times 2^m$ matrix can be distinct. However, matrices that represent circuits often have many identical rows. For example, in the PTM of an ideal n-input *NAND* gate all but one of the rows are identical (set $p = 0$ in Figure 1). This suggests that circuit matrices can be significantly compressed.

In [3] the authors describe the encoding of a matrix using ADDs. Recall that a BDD is a directed acyclic graph representing $f(x_0,x_1,x_2,\ldots x_n)$ with root node x_0. The subtree formed by the outgoing edge labeled 0 represents the cofactor $f_{x_0'}(x_1\ldots x_n)$, and the subtree formed by the outgoing edge labeled 1 represents the cofactor $f_{x_0}(x_1\ldots x_n)$. Boolean constants are represented by terminal nodes. ADDs are variants of BDDs where terminal nodes can take on any real value. ADDs can also have multiple terminals.

In the ADD encoding of a matrix, the nodes represent decisions on row and column variables, and terminals represent values in matrix entries. The entries themselves are represented by paths starting from the root node and ending in a terminal. The same path can encode several entries if variables are skipped in a path. The input variables are queried in a pre-defined order, and this facilitates reductions by using the same node for identical sub-matrices. We use the QuIDDPro library [12] to encode PTMs into ADDs and also to perform operations on PTMs. QuIDDPro includes the CUDD library and uses interleaved row and column variable ordering. This ordering facilitates fast tensor products and matrix multiplications — key operations in quantum-mechanical simulations for which QuIDDPro was designed.

4.2 Handling Non-Square Matrices

All matrix algorithms for ADDs that we are aware of assume square matrices, but can represent non-square matrices using zero padding [3, 4, 12]. A non-square matrix has fewer row variables than column variables or vice versa. However, any missing variable in a standard ADD is interpreted as marking replicated matrix entries. In other words, the entries of the matrix for both values of the variable are identical. To prevent this effect, missing rows or columns can be explicitly padded with zeros.

Matrix multiplication and addition are compatible with zero padding [3], however the tensor product is not. When two padded matrices A and B are tensored together, the result will have spurious rows of zeros which are carried over from the zero-padding of B.

Example 2 The equation below is an example of an ideal *NOT* gate tensored with an ideal zero-padded *NAND* gate.

$$\underset{(NOT)}{\begin{bmatrix} 0&1\\ 1&0 \end{bmatrix}} \otimes \underset{(NAND2)}{\begin{bmatrix} 0&1&0&0\\ 0&1&0&0\\ 0&1&0&0\\ 1&0&0&0 \end{bmatrix}} = \begin{bmatrix} 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&1&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0 \end{bmatrix}$$

This illustrates the incorrect results obtained from tensoring padded matrices. Columns 3 and 4 erroneously consist entirely of zeros which are carried over from the zero-padding of the *NAND* matrix.

We develop two approaches to reconcile tensor products with zero-padding: dummy outputs and the permutation matrices. In the first approach, we add dummy outputs to a gate

so as to equalize the number of inputs and outputs. This makes the gate PTM square thereby eliminating the problem with tensor products.

Dummy outputs can be subsequently removed by multiplying the result by a fanin-gate PTM, such as the one on the right for a 2-input 1-output fanin gate. In general, a fanin gate is the transpose of the fanout gate introduced in Section 3. The fanin gate shown here is itself zero-padded and so in essence it moves desired outputs to the left corner of the matrix and the all-zero columns to the right. ITMs for fanin gates can be directly constructed as ADDs by first creating an appropriately-sized identity matrix and then *abstracting* column variables corresponding to the outputs we wish to retain, as explained below.

$$\begin{bmatrix} 1&0&0&0\\ 1&0&0&0\\ 0&1&0&0\\ 0&1&0&0 \end{bmatrix}$$

Definition 4 *The* abstraction *of signals s and t from matrix M is matrix M_0 with entries $p(k|j) = \sum_{all\ x}\sum_{all\ y} p(s = k,x|t = j,y)$. Here the $p(k|j)$ is the sum over all possible values of the remaining input and output variables of M.*

This operation generalizes the well-known existential abstraction of a single or multiple Boolean variables, but relies on arithmetic addition of matrix entries instead of the Boolean disjunction of cofactors. Its implementation in terms of ADDs is similar.

The alternative, *permutation matrix* method shifts spurious columns of zeros to the end of the tensor-product matrix. This can be expressed as multiplication by a permutation matrix acting as a column permutation, but such permutations have size up to 2^n (for n inputs) and are impractical if specified explicitly. Fortunately, we discovered a decomposition of *zero-tracking* permutation matrices into products and tensors of smaller matrices, which dramatically reduces the runtime.

Suppose we are calculating $A \otimes B$ for two matrices A and B with dimensions $2^p \times 2^p$ and $2^q \times 2^q$ after zero-padding. Spurious rows or columns of zeros will appear only if the actual dimensions of B are smaller. To this end, suppose that B has actual dimensions $2^n \times 2^q$, $n < q$. Then the *zero-tracking row permutation* R is defined as $R = r \otimes I_n$ where I_n is a $2^n \times 2^n$ identity and r is a $2^{p+q-n} \times 2^{p+q-n}$ matrix where i-th row has a 1 in entry $(i,\ i2^{q-n} \bmod 2^{p+q-n})$ and zeros elsewhere. The zero-tracking column permutation matrix is defined similarly; see matrix R in Example 3.

A key observation is that R can be decomposed into a product of wire permutations and tensored with identity matrices according to

$$R = \Pi_{k=n}^{q-1} rperm(I_{p+q-k}, \sigma_k) \otimes I_k \qquad (2)$$

where $\sigma_k(i) = (i-1) \bmod (p+q-k)$ Wire permutations are special cases of row variable permutations (see Definition 3). Since there are exponentially fewer row variables than rows (similarly for columns), these special-form matrices can be encoded much more compactly.

Example 3 To address the problem noted in Example 2 we need the zero-tracking permutation matrix R, shown below.

$$\underbrace{\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix}}_{(R)} = \underbrace{\begin{bmatrix} 1&0&0&0\\ 0&0&1&0\\ 0&1&0&0\\ 0&0&0&1 \end{bmatrix}}_{(rperm)} \otimes \underbrace{\begin{bmatrix} 1&0\\ 0&1 \end{bmatrix}}_{(I_1)}$$

Here, matrix $rperm$ is a 4×4 identity (I_2) with its 1-bit column variables swapped (columns 01 and 10 are swapped).

Intuitively, for a circuit with n row variables $r_0, r_1 \ldots r_n$ this corresponds to moving r_n to r_{n-1}, r_{n-1} to r_{n-2} and so on. This method is efficient because the $rperm(I_{p+q-k}, \sigma_k)$ gates are row variable permutations on the identity matrix and I_k is a k-variable identity matrix. For any n, $I_n = I_{n-1} \otimes I_1$, and this recursive calculation of I_n takes linear time in n [12]. The row variable permutation also takes $O(n)$ space because a matrix with 2^n rows has only n row variables. We note that decomposition of permutations into tensor products was studied in [5]. However, that work did not use variable permutations while we do not use conjugated tensor products.

4.3 Evaluation Procedure

PTM evaluation proceeds by level, from the primary outputs to the primary inputs; see Figure 3. First the primary outputs are stored in the list called *CurrSigs*. The gates that output these signals are tensored together to form the PTM for *CurrLevel* and are removed from *CurrSigs*. The *CurrLevel* PTM is then multiplied by the circuit PTM to form the new *CurrCircuit* PTM. This process continues until the *CurrSigs* are identical to the primary input signals and avoids tensoring gate PTMs together until necessary.

The intermediate level PTMs can be discarded after they are multiplied into the circuit PTM. This is important for the scalability of a circuit because tensoring two PTMs of size $2^n \times 2^n$ and $2^m \times 2^m$ produces a result with $2^{n+m} \times 2^{n+m}$ entries. So suppose a level has n inputs and m outputs, then PTM is given by a $2^n \times 2^m$ matrix. If these can be stored seperately as individual gates with one output per gate then the size is only $m2^n$. This is an exponential decrease in the space complexity.

```
CurrSigs = primary outputs
while (CurrSigs ≠ primary inputs)
  for j = 0:CurrSigs.size()
    CurrLevel = CurrLevel ⊗ gate_lookup(CurrSigs[j])
    zero_track(CurrLevel)
    remove_redundancy(CurrLevel)
  end
  CurrCircuit = CurrCircuit * CurrLevel
  CurrSigs = CurrCircuit.inputs
end
```

Figure 3: Pseudo-code for our PTM evaluation method.

In Figure 3 the function $gate\_lookup(S)$ returns the PTM for the gate which is the source for signal S. If the source gate has multiple outputs then the desired signal is abstracted. If S is a primary input or needed in a future iteration (this is looked up in a table), the function returns the identity gate with S as the input thereby propagating the input to the previous level. At the termination of the procedure the current signals should include all primary inputs and only primary inputs.

To improve the efficiency of PTM evaluation in practice, we introduce two additional ADD operations: one for removing redundancy in input signals and one for reliability calculation. In place of fanout gates, described in Section 3, we use an operation called *redundancy removal*. By removing each duplicated input signal, instead of adding a fanout gate, the PTM size is cut in half. Consider the case where inputs i_0 and i_2 are identical. In this case, all PTM

Circuits	Characteristics				Performance, $p = 0$		Performance, $p = .05$		Reliability, $p = .05$	
	gates	inputs	outputs	width	time(s)	memory (MB)	time(s)	memory(MB)	two-way	one-way
C17	6	5	2	5	.057	.003	.076	.003	.846	.880
mux	6	21	1	23	6.13	3.16	6.13	2.51	.907	.939
z4ml	8	7	4	20	1.74	1.54	2.22	1.59	.67	.817
x2	12	10	7	23	18.8	7.91	35.3	9.03	.15	.099
parity	15	16	1	23	.350	.129	.350	.144	.602	.731
pcle	16	19	9	16	12.7	3.61	74.9	24.2	.573	.657
decod	18	5	16	13	2.17	1.72	56.9	11.8	.000	.000
cu	23	14	11	23	5.12	2.53	93.87	10.0	.461	.579
pm1	24	27	17	27	24.6	6.95	7169	160	.375	.596
9symml	44	9	1	37	362	612	1758	1200	.327	.534
xor5	47	5	1	19	19.3	4.42	1337	57.3	.067	.071

Table 1: Performance of PTM evaluation and reliability of circuits with gate error $p = 0.05$.

rows whose indices have different bits 0 and 2 do not represent observable behavior and should be discarded, including row $6 = 110_2$. The redundancy removal is implemented as a variation of the *abstraction* operation (see Definition 4). In *abstraction* entries corresponding to the variables being eliminated are summed together; for redundancy removal, one of the two rows corresponding to the redundant signal is chosen. This process can be repeated for more redundancy.

The *reliability* operation determines the probability of correctness of a circuit by comparing the ITM with the PTM.

Definition 5 *The* reliability *of a circuit c with PTM M and ITM J is given by* $Rel(c) = \Sigma_{J(i,j)=1} p(j|i)p(i)$. *Recall that* $p(j|i)$ *is found in the* (i, j)*th entry of M. In the case that all inputs are equally likely,* $Rel(g) = \frac{1}{2^n}\Sigma_{J(i,j)=1} p(j|i)$ *where n is the number of inputs to c.*

4.4 Empirical Results

Results from calculation of circuit ITMs and circuit PTMs are shown in Table 1 for smaller LGSynth 91 and LGSynth 93 benchmarks with independent uniform distributions on all primary inputs. These simulations were run on a Linux workstation with a 3GHz Pentium 4 processor. In our experiments CPU time was limited to 24 hours. The runtimes and memory requirements are sensitive to the width of a circuit, i.e., the largest number of signals at any level. This determines the size of the tensor products and permutations/fanin gates for zero-tracking in tensor products. Empirically, circuits with widths of around 40 signals can be evaluated efficiently. Memory is not a bottleneck in our experiments.

Table 1 gives the overall reliability of the circuits for gate error probabilities of 0.05 and also one-way gate errors of 0.05. In CMOS gates, an erroneous output value 0 is more likely than an erroneous value 1 because SEUs typically short-circuit power to ground. PTMs can encode this bias easily because the error probability depends on the input value. Relevant empirical results are given in the "one-way" column in Table 1. Note that circuits with a high output-to-input ratio, such as decod.blif, tend to magnify gate errors at fanout stems and therefore have lower reliability.

PTM evaluation for $p = 0.05$ takes more memory and runtime because less compression is possible. Ideal matrices have large blocks of 0s which makes for more identical submatrices to compress. When PTMs with errors are composed, there is greater variety in the matrix entries yielding less compression.

5 Applications

PTM evaluation has a wide range of applications. Below we describe two sample applications that we have implemented.

5.1 Gate Susceptibility

Replication is often used to increase the reliability of a circuit e.g. TMR (triple modular redundancy). In [8] the authors show that in some circuit applications it may not be necessary to replicate the entire circuit. Only certain gates which are especially susceptible to error are chosen for replication and this increases the reliability of a circuit with relatively little area overhead. According to their heuristic, errors are injected into gates and then a sample of input vectors is applied in order to assess the probability of error at the primary outputs. Since the number of input vectors is exponential in the number of inputs to the circuit, this process can have high time complexity. Further, it can be inaccurate because a few inputs which are not included in the sampled set can lead to a drastic increase in the gate susceptibility.

PTMs can provide an accurate measure for the susceptibility of a gate using. The *observability* of a gate g in a circuit C can be thought of as the observability of its output signal. It is defined as the probability of output error given that gate g has error probability p on all inputs. This value is obtained by calculating $Rel(C)/p$ where $Rel(C)$ is reliability after insertion of an error into g. The gates with highest observability can be regarded as the most susceptible to probabilistic errors.

Since PTM calculations implicitly include all input vectors, sampling is unnecessary. Observability can be computed by introducing an appropriate error into gate PTMs and then evaluating the circuit PTM. This method can handle various error types, e.g., one-way errors.

Circuit	Orig.	Top 3	Improved	Top 5	Improved
C17	.864	.959	11.0 %	.980	13.4 %
mux	.907	.974	7.39 %	.985	8.6 %
parity	.603	.637	5.64 %	.666	10.4 %
xor5	.047	.068	46.2 %	.070	50.5 %
pm1	.375	.429	14.4 %	.469	25.1 %

Table 2: Improvement in reliability after increasing robustness of the 3 and 5 most susceptible gates.

Table 2 illustrates gate susceptibility calculations in some of the standard benchmarks. The most susceptible gates are hardened by decreasing the probability of error by a factor of 10 while retaining the error probability of .05 in all

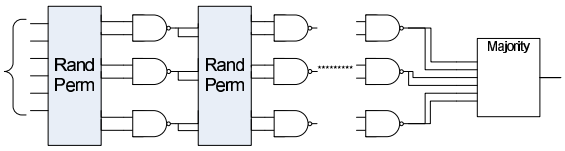

Figure 4: Von Neumann's NAND-multiplexing scheme.

other gates. For many circuits increasing the robustness of just a few gates can improve circuit reliability significantly. Note an exact calculation can can identify signals with zero observability which implies that the circuit contains redundancy. This cannot be done with an approximate method.

5.2 NAND-Multiplexing Analysis

A classic application of PTMs is the evaluation of fault-tolerant architectures. The advent of nanotechnology has renewed interest in building reliable circuits from unreliable components because nanodevices are poorly controlled but available in large quantities, justifying computing techniques with high levels of redundancy, such as NAND-multiplexing and Cascading TMR which originated in [13].

In NAND-multiplexing, each unreliable signal is replicated N times. Then a set of NAND gates, each of which takes two of the N redundant signals as inputs, is used as a simple majority function. Some of these NAND gates may produce the wrong output due to an unfortunate combination of inputs. For instance, if 25% of the N replicated signals are 0 and 75% are 1, then 1 is most likely the correct value; however, some NAND gates may have both their inputs from the erroneous 25%. In order to compensate for this effect, the outputs of the NAND gates themselves are replicated and then fed into a "random permutation" gate. The outputs of the random permutation gates enter another series of NAND gates. These gates are likely to have fewer 0's since there are relatively few combinations which give the wrong value.

In order to calculate the reliability of N iterations of NAND-multiplexing, several types of errors need to be taken into account. These include

- Errors in the gates including the NAND gates.
- Errors due to combinations of inputs.
- Pseudo-randomness in the permutation which biases the combinations of inputs applied to NAND gates.

In [6], this architecture is evaluated using analytical techniques. We can automate this type of analysis while relaxing some of the assumptions such as identical probabilities of error in gates, independence of input signals, and perfect randomness. Using PTM evaluation, we can input the specific probability of error and function of each of the gates, including the permutation gate analysis thereby automating the analysis. This will also take into account all the ways in which errors can cancel.

Table 3 shows results from a NAND-multiplexing instance with four replicated inputs, all of which have correct value 1 and are erroneous with probability 0.2. They are then passed through a number of NAND-multiplexing levels. Table 3 shows the probability that errors are corrected (signals are all returned to an identical 1 state). The multiplexing units themselves have probabilities of error for their constituent gates and also for the random-permutation unit. Table 3 shows that depending on the gate error probabilities, increasing the number of multiplexing stages can either increase or decrease the amount of error cancellation. This behavior is also not monotonic, thereby making analytical calculations difficult. Results for eight input replications show a similar trend.

Gate error	Number of NAND-MUX levels				
	2	4	6	8	10
.05	.8075	.7780	.7470	.7190	.5740
.02	.9160	.9144	.9074	.9005	.8175
.005	.9741	.9795	.9789	.9784	.9544

Table 3: NAND-multiplexing with inputs replicated 4 times.

6 Conclusion and Ongoing Work

In this paper we investigated the PTM framework for representing errors at the gate level, stressing its use in analyzing soft errors in logic. We provided an implementation for exact reliability calculations which used matrix operations to calculate circuit PTMs. The implementation, which compresses circuit-matrices using ADDs, shows that exact reliability measures can be calculated efficiently for larger circuits than previously thought.

PTM evaluation has a wide variety of applications. Exact reliability calculations are useful for analyzing fault-tolerant architectures and in calculating gate error susceptibility. Other applications include measuring the testability of signals under transient errors, and reliability-driven logic restructuring where parts of circuits are re-synthesized.

Our ongoing work promises to extend PTM evaluation to handle sequential logic blocks in addition to combinational logic blocks. This involves breaking circular dependencies in calculating conditional error probabilities. To increase the scalability of PTMs, partitioning based on logic cones, which is used in pseudo-exhaustive testing [11], can be used to break larger circuits into small components. The PTM for each component is calculated, then a path tracing method can be used to compute the result for specific instantiations of a circuit. This can increase the accuracy and speed up approximate circuit reliability calculations.

Acknowledgments. This work was supported by the National Science Foundation, the U.S. Department of Energy and the Gigascale Systems Research Center. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements of funding agencies. We also thank Ketan Patel of QualComm for helpful discussions.

References

[1] D. Alexandrescu, L. Anghel, M. Nicolaidis, "New Methods for Evaluating the Impact of Single Event Transients in VDSM ICs " *IEEE Symp. on Defect and Fault Tolerance in VLSI Sys.,* 2002, pp. 99-107.

[2] R.I. Bahar, J. Mundy and J. Chan, "A Probabilistic Based Design Methodology for Nanoscale Computation", *ICCAD*, 2003, pp. 480-486.

[3] R.I. Bahar et al., "Algebraic Decision Diagrams and their Applications," *J. of Formal Methods in Sys. Design* 10, no.2/3, April-May 1997, pp. 171-206.

[4] E. Clarke et al., "Multi-Terminal Binary Decision Diagrams and Hybrid Decision Diagrams," in T. Sasao and M. Fujita, eds, *Representations of Discrete Functions*, Kluwer, 1996, pp. 93-108.

[5] S. Egner, M. Püschel, and Th. Beth, "Decomposing a Permutation into a Conjugated Tensor Product," *Intl. Symp. Symbolic and Algebraic Comput.* (ISSAC) 1997, pp. 101-108.

[6] J. Han, P. Jonker,"A System Architecture Solution for Unreliable Nanoelectronic Devices,"*IEEE Trans. on Nanotechnology*, vol. 1, December 2002 pp. 201-208.

[7] V. L. Levin,"Probability Analysis of Combination Systems and their Reliability,"*Engin. Cybernetics*, no 6. Nov-Dec. 1964, pp. 78-84.

[8] K. Mohanram and N. A. Touba, "Cost-Effective Approach for Reducing Soft Error Failure Rate in Logic Circuits," *ITC*, 2003, pp. 893-901.

[9] K.N. Patel, J.P. Hayes, and I.L. Markov, "Evaluating Circuit Reliability Under Probabilistic Gate-Level Fault Models," *IWLS,* May 2003, pp. 59-64.

[10] P. Shivakumar, M. Kistler, et. al, "Modeling the Effect of Technology Trends on Soft Error Rate of Combinational Logic" *Intl. Conf. on Dependable Systems and Networks*, 2002, pp. 389-398.

[11] R. Srinivasan, S.K. Gupta and M. A. Breuer, "An Efficient Partitioning Strategy for Pseudo-Exhaustive Testing" *DAC*, 1993, pp. 242-248.

[12] G. F. Viamontes, I. L. Markov and J. P. Hayes, "Improving Gate-Level Simulation of Quantum Circuits", *Quantum Information Processing*, vol. 2(5), October 2003, pp. 347-380. `http://arxiv.org/abs/quant-ph/0309060`

[13] J. von Neumann,"Probabilistic Logics & Synthesis of Reliable Organisms from Unreliable Components," *Automata Studies*, 56, pp. 43-98.