An Efficient Methodology to Evaluate Nanoscale Circuit Fault-tolerance Performance based on Belief Propagation

Huifei Rao, Jie Chen+, Vicky H. Zhao, Woon Tiong Ang, I-Chyn Wey* and An-Yeu Wu*
Electrical and Computer Engineering Department, University of Alberta, Canada
+National Institute of Nanotechnology, Canada
**Graduate Institute of Electronics Engineering, and Department of Electrical Engineering, National Taiwan University, Taiwan*

***Abstract*— As silicon circuits quickly approach their physical limitations, researchers are actively looking for novel building blocks to develop nanocircuits. However, future nanoelectronic circuits are more error-prone than conventional CMOS designs because of their self-assembly design. To help design fault-tolerant nanoscale circuits, new circuit design and testing tools are needed. In this paper, an efficient methodology to evaluate nanoscale circuit fault tolerance based on Belief Propagation (BP) algorithm is proposed. Compared with existing approaches, the BP algorithm is more efficient in terms of memory requirements and CPU times. The proposed methodology can be easily run on multiple CPUs to achieve parallel processing and thus further reduces simulation time.**

I. INTRODUCTION

In nanoscale devices and circuits, two types of errors are unavoidable, i.e., *hardware faults* and *signal faults* [2], [3]. In order to make nanoscale systems reliable, the design of fault-tolerant architectures is necessary. Building fault tolerant design using unreliable components was initially proposed by Von Neumann [4]. He proposed the use of universal gates (such as NAND and majority logic gate) as a primitive building block and used multiplexing technique to improve fault-tolerance. However, the hardware cost for building Von Neumann's multiplexing is too high. This technique was extended in [5] to a rather low-degree redundancy to reduce hardware usage. In [6], bifurcation theory and its associated geometrical representation were used to analyze the Markov multiplexing system. Their work showed that the two modes (uni-modal and bi-modal) and the median of the stationary distribution are critical to characterizing the reliability of the system. They also illustrated how the NAND-multiplexing technique can lead to system reliability in spite of large gate error probability, while the cost of redundancy is kept moderate.

To date, many nanoscale fault-tolerant circuit design methodologies have been proposed. A probabilistic-based matrix model has been reported [1] and verified [2] to deal with faults in a nanoscale system. This matrix model is easy to use and is suitable for employment in CAD tools. However, both CPU time and memory requirements grow exponentially as the number of inputs and outputs increases. In [3], algebraic decision diagrams were employed to improve the efficiency of the matrix model, but the memory requirements are huge for complicated circuits. In this paper, we propose a new methodology to evaluate error distribution within circuits based on belief propagation. As we know, a circuit's fault-tolerance behavior very much depends on its topology [2]. For instance, there are different ways to design adders, including ripple adders, look-ahead adders, etc. Our error distribution results can help circuit designers determine which design is the most fault-tolerant among various designs. The knowledge can also help us find critical paths from circuit inputs to outputs that are vulnerable to hardware and signal faults.

The paper is organized as follows. Section II introduces the BP algorithm and the challenge of applying it in our design. In Section III, we use an example to show how to apply the BP algorithm to evaluate performance of digital circuits. Section IV shows the simulation results and the comparisons between our design and other methods. We finally made the conclusion in Section V.

II. BELIEF PROPAGATION

Belief Propagation was initially proposed by Judea Pearl [7] as a dynamic programming approach to solve conditional probability queries in graphical models. Belief propagation has a wide variety of applications in computer vision, speech recognition and error correction codes.

Let us take a simple example introduced in [8] to illustrate how the scheme can be applied to solve basic problems that we are interested in.

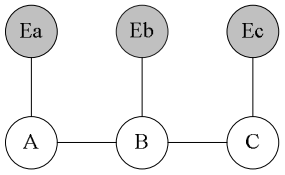

Figure1. A simple problem that can be solved by BP algorithm

As shown in Figure1, the problem is to infer the states of three hidden nodes (A, B, C) given the states of three observed nodes (E_a, E_b, E_c). The probability distribution is given by:

$$P(A,B,C \mid E_a, E_b, E_c) = \frac{1}{Z} e^{-J(A,B,C,E_a,E_b,E_c)}$$

where Z is a normalization factor and

$$J = J_1(A,B) + J_2(B,C) + J_3(A,E_a) + J_4(B,E_b) + J_5(C,E_c)$$

is the energy function. The probability function can be factored as

$$P = \frac{1}{Z}\psi_1(A,B)\psi_2(B,C)\psi_3(A,E_a)\psi_4(B,E_b)\psi_5(C,E_c)$$

where $\psi_i = e^{-J_i}$.

Assume our task is to find the probability that node A is in a particular state given all the evidence Ev=(E_a, E_b, E_c), that is, P(A=a | Ev). Intuitively, we can use exhaustive enumeration to solve this problem, e.g. $P(A=a \mid Ev) = \sum_{b,c} P(A=a, B=b, C=c \mid Ev)$. However, the complexity of this method is exponential function of the number of hidden nodes.

In [7], Pearl derived an efficient local message-passing method (BP algorithm) to calculate the posterior probability of a hidden variable given the observed variables (the evidence) for single connected Bayesian network—a network in which

there is only one path between any two nodes. The BP algorithm consists of simple local updates and can be executed in parallel. Pearl proved that this algorithm is guaranteed to converge to the correct answers for any single connected Bayesian network.

As described in our previous work [2], the fault-tolerance performance of a circuit can be evaluated by its System Error Rate (SER). The calculation of SER could be decomposed into the posterior probability inferring tasks described previously. This fact gave us the inspiration to evaluate the fault-tolerance performance of a circuit using the BP algorithm. To demonstrate how belief propagation can help efficiently calculate SER and evaluate the fault-tolerance performance of nanoscale circuits, we use a simple XOR with four gates as example. The circuit schematics is shown in Figure 2.

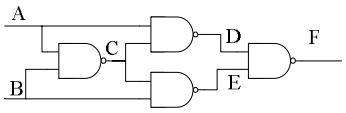

Figure2. An XOR gate

The SER of the XOR circuit in Figure2 is:

$$SER = 1 - \sum_{(a,b)=(0,0)}^{(1,1)} P(F = f \mid A = a, B = b) \bullet P(A = a, B = b) \qquad (1)$$

where f is the output value corresponding to the inputs (A,B) in the truth table. $P(A = a, B = b)$ is the prior distribution which we already know. $P(F = f \mid A = a, B = b)$ is the posterior probability we need to infer. The calculation of SER can be decomposed into the computations of conditional probabilities, which can be solved efficiently by BP.

However, in order to apply BP, there is one challenge we need to address. As Pearl has pointed out, the BP algorithm is not guaranteed to converge or converge to an accurate value in a multiple connected graph because of "double counting"—a case that the same evidence is passed around the network multiple times and mistakenly treated as a new evidence [7][8]. For most practical circuits, their corresponding Bayesian graphs are multiply-connected graphs due to the reconvergent fanout. For example, in Figure2, node C fanouts to two different signals, D and E, which converge to F in the later stage. As a result, these connections form a loop among C, D, E and F (see Fig. 4). We therefore need an algorithm which is able to handle loopy graphs. We will introduce a solution and discuss its application in the next section.

III. Circuit Fault-Tolerance Evaluation Using Probability Propagation in Trees of Clusters (PPTC)

PPTC algorithm was developed by Lauritzen and Spiegelhalter and refined by Jensen. As shown in Figure 3, PPTC first converts the belief network into a cluster tree structure. Probabilities are then computed by propagating messages in the tree structure. In this section, we will use the circuit in Figure 2 as an example to show the procedures for calculating circuit SER using the PPTC algorithm. We will follow [9] and use the notations and definitions there.

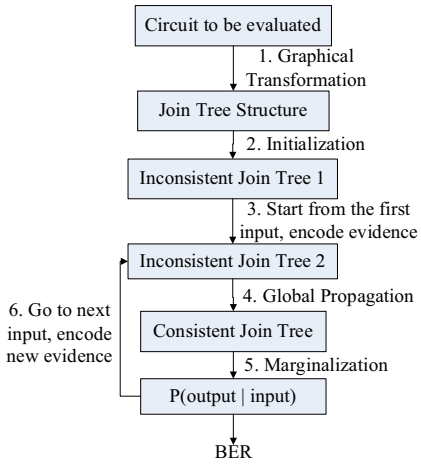

Figure3. Block Diagram of PPTC

1) Graphical Transformation

1.1) **Build Bayesian graph:** We first create a node for each signal in the circuit and then connect every input-output pair of each gate using an arrow. For example, in the XOR gate, A and B are parents of C, so we connect AC and BC using two arrows. The Bayesian graph of the XOR gate in Figure2 is shown in Figure4.

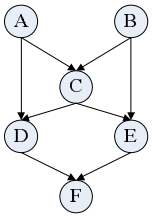

Figure4. The Bayesian graph

1.2) **Construct Moral Graph:** Drop all the directions in the Bayesian graph. For each node in the graph, connect each pair of its parents. For example, A and B are parents of C, we connect A and B with a dotted-line as shown below.

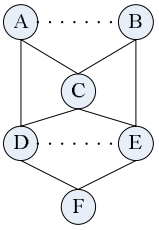

Figure5. The Moral graph

1.3) Triangulation

A graph is triangulated if and only if every cycle (with length>=4) contains an edge that connect two non-adjacent nodes in the cycle. Details of the triangulation process are described in [9]. We only show the result of the triangulated graph of our example. As we can see, a dotted-line is added between B and D to make the graph triangulated in figure 6. It should be noted that the triangulation process is not unique, which means that the final triangulated graph might be different due to different choices we make in the triangulation process. As long as we follow the criterion described in [9] during the process, it is guaranteed that the final joint tree that we build will be optimal.

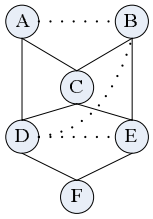

Figure6. The triangulated graph

1.4) Identify Cliques

A clique is a sub-graph that is complete and maximal. Complete means that every pair of nodes in the clique is connected. Maximal means that the clique is not contained in a large and complete subgraph. As indicated in [9], the cliques can be identified during the triangulation process. In our example, the cliques are: {A,B,C,D}, {B,C,D,E}, {D,E,F}.

1.5) Build Join Tree: A join tree (or a junction tree, a cluster tree in the literature) is a graph which is composed of clusters, sepsets and their interconnections. Clusters are the cliques we got in step1.4, sepsets are intersections of adjacent clusters. The constraints that a joint tree must satisfy and the process of building a joint tree are described in [9]. The joint tree of our example is shown in Figure7. The ellipses represent clusters while the rectangles represent sepsets.

Figure7. The joint Tree

2) Initialization: For each cluster X in the joint tree in Figure 7, it contains two potential arrays, $\phi_X^{Initial}$ and $\phi_X^{current}$, where potential is the joint probability P(X, Ev) in our example (Ev is the evidence). The first array is used to store initial potentials, and the second array is used in the following step for the Global Propagation. Here, we will initialize the first array as follows.

I) For each cluster X, set each $\phi_X^{Initial}(x)$ to 1.

II) Assign each variable V a cluster X which contains V and V's parent $\prod_V$, then multiply $\phi_X^{Initial}$ by P(V| $\prod_V$).

In our example, the initialization results are:

$\phi_{ABCD}^{Initial}$ =P(C|A,B)P(D|A,C); $\phi_{BCDE}^{Initial}$ =P(E|B,C);

$\phi_{DEF}^{Initial}$ =P(F|D,E).

3) Encode evidence: In this step, we will extract the evidence based on current input and output values and encode this information into a cluster potential array.

I) Extract the evidence.

For example, the first input is (A,B)=(0,0), its output is F=0, the conditional probability we want to infer is

$$P(F=0|A=0,B=0)=\frac{P(A=0,B=0,F=0)}{P(A=0,B=0)}=P(A=0,B=0,F=0)\ ,$$

given P(A=0,B=0)=1.The evidence Ev=(A=0,B=0), what we need to calculate is the potential $\phi(F=0)$

II) For each cluster X, set $\phi_X^{current}=\phi_X^{Initial}$. For each sepset R, set $\phi_R^{new}\equiv 1$.

III) Encode observation V=v as Λ_V, identify a cluster X that contains V and multiply ϕ_X by Λ_V.

For example, $\Lambda_A=(1,0)$, $\Lambda_B=(1,0)$, cluster ABCD contains A and B, so $\phi_{ABCD}^{current}=\phi_{ABCD}^{current}\bullet\Lambda_A\bullet\Lambda_B$.

4) Global Propagation: Now, we have initialized the joint tree potentials and encode the evidence into the cluster potentials. We need to perform the global propagation to make them locally consistent. In general, the global propagation is executed in three steps:

I) Choose an arbitrary cluster X.

II) Unmark all clusters. Call Collect-Evidence(X).

III) Unmark all clusters. Call Distribute-Evidence(X).

In the above, Collect-Evidence is the procedure for propagating messages in the direction toward cluster X, and Distribute-Evidence is the procedure for propagating messages in the opposite direction.

In our application, however, what we are interested in is the potential of the output variable F given the evidence. We do not have to make all clusters in the tree locally consistent. Instead, we only need to make one cluster containing F locally consistent. We can therefore reduce three steps into two steps:

I) Choose a cluster X which contains the last output variable, e.g. Cluster DEF in our example.

II) Unmark all clusters. Call Collect-Evidence(X).

The procedure of Collect-Evidence(X) is:

I) Mark X

II) Call Collect-Evidence recursively on X's unmarked neighboring clusters

III) Pass a message from X to the cluster which invokes Collect-Evidence(X)

A single message passing procedure (between two adjacent clusters X and Y with sepset R) in step3 of Collect-Evidence is :

I) $\phi_R^{old}=\phi_R^{new}$

II) $\phi_R^{new}=\sum_{X\backslash R}\phi_X^{current}$, where X\R means the subset of X after eliminating the variables of R from X. For example, if X={A,B,C,D}, R={B,C,D}, then X\R={A}

III) $\phi_Y^{current}=\phi_Y^{current}\bullet\frac{\phi_R^{new}}{\phi_R^{old}}$

5) Marginalization: After we have the consistent cluster potential $\phi_{DEF}^{current}$, we can calculate the potential $\phi(F=0)$ by marginalization, $\phi(F=0)=\sum_{F=0}\phi_{DEF}^{current}$

6) Handle next input: Move to the next input, get the corresponding output and repeat step3~step5 until we reach the last input ($AB=11$ in our example). Now, we can use Formula (1) in section II to compute circuit SER.

Notes that step1 and step2 only need to be executed once in individual circuits. However, step3 to step6 need to be executed repeatly for each circuit input.

IV SIMULATIONS AND COMPARISONS WITH OTHER METHODS

Since all computations are executed locally, the memory requirement is small. Three major steps that require most of memory are:

I) The Conditional Probability Table of various logic gates.

II) The potential arrays of each cluster ($\phi_X^{initial}$ and $\phi_X^{current}$).

III) The potential arrays of each sepset (ϕ_R^{old} and ϕ_R^{new}).

To demonstrate the performance of the proposed algorithm, we use 4-bit, 8-bit and 16-bit ripper carry adders (RCA) and conditional select adders (CSA) [10], respectively, as benchmark circuits. We run our simulations on a Pentium IV 3.0G desktop with 1G memory for different types of adders with different input bits. The results are shown in Table 1.

circuit	#of input	# of output	# of gate	Memory (kB)	CPU Time (s)
4-bit RCA	9	5	56	692	0.015
4-bit CSA	9	5	58	702	0.11
8-bit RCA	17	9	112	704	4.754
8-bit CSA	17	9	118	756	30.969
16-bit RCA	33	17	224	716	637371
16-bit CSA	33	17	238	808	4337254

Table 1. Simulation results

We can see from Table 1 that the memory cost for all the circuits in simulation is less than 1MB and does not depend on circuit size very much.

We also compare the performance of the BP-based algorithm with prior work. In [1],[2], we calculated circuit SER directly from the ensemble-dependent matrix. This matrix model is easy to use, but the matrix size grows exponentially with the number of inputs and outputs. It is not practical to compute the error distribution of large-scale circuits. For a 4-bit CSA, the matrix size is several hundred MBs. For an 8-bit CSA, the matrix size exceeds 1TB, which is beyond the capability of current computers. The matrix model was improved in [3] to handle larger circuits more efficiently. Compared to this improved matrix method, our BP method still shows significant advantage. For example: the circuit 9symm1 (9 inputs, 1 outputs, 44 gates) in [3] has comparable size and complexity with the 4-bit adders in our simulations. It took 1758 seconds and 1200MB to compute SER of this circuit in [3]. Another example is circuit pm1 (27 inputs, 17 outputs, 24 gates), which took 7169 seconds and 160MB to calculate its SER in [3]. In our simulations, the 16-bit RCA costs about 637371 seconds and 0.716MBs. For a fair comparison with our 16-bit RCA adder that has 33 inputs, we divide 637371 by 2^6 (which is the difference between the number of inputs of 16-bit RCA and pm1). With our proposed approach, even though the 16-bit RCA has much more gates than pm1, the proposed algorithm uses approximately the same amount of time to calculate the SER of the 16-bit RCA as that required to calculate the SER of circuit pm1 using the approach in [3]. Overall, our method demonstrates better performance than that in [3].

The CPU time consumed in calculating each P(output | input) is short due to the use of belief propagation, which eliminates many intermediate computations. However, there is a total of 2^m (m is the number of circuit inputs) conditional probabilities to be computed. The overall CPU time is still large as circuit size grows. This issue remains as a challenge for our future research.

The total CPU time can be reduced in two ways, 1) the computations can be easily distributed to multiple processors so that the computation time can be reduced by the number of processors. For instance, in our XOR example, we can calculate the conditional probabilities of inputs 00 and 01 in a computer, and calculate the conditional probabilities of inputs 10 and 11 in another computer, then combine their results to get the final SER. 2) We can group the gates in the circuit into submodules and then use the BP algorithm based on these submodules instead of the original logic gates.

V CONCLUSION

Structural and signal errors are unavoidable in nano devices. In this paper, we use the BP algorithm to compute error distributions in circuits of different topologies. Based on circuit SER, we can compute error probability at each node and thus we can evaluate overall circuit fault-tolerance performance. Our method is efficient in terms of computation complexity. The program built on our method can be easily deployed on multiple processors for parallel computing. Our future research focuses on efficiently handling the computation for larger circuit, such as 32-bit and 64-bit adders.

The authors would like to acknowledge the National Science Foundation of US and Natural Sciences and the Engineering Research Council of Canada for the funding support.

REFERENCES

[1] K. N. Patel, I. L. Markov, and J. P. Hayes, "Evaluating circuit reliability under probabilistic gate-level fault models," in IEEE International Workshop on Logic and Synthesis, 2003.

[2] Huifei Rao, Jie Chen, Vicky H. Zhao, Woon Tiong Ang, Changhong Yu, Huaixiu Zheng, I-Chyn Wey, An-Yeu Wu, "Ensemble Dependent Matrix Methodology for Probabilistic-based Fault-Tolerant Nanoscale Circuit Design," IEEE International Symposium on Circuits and Systems 2007

[3] Smita Krishnaswamy, George F. Viamontes, Igor L. Markov, John P. Hayes, "Accurate Reliability Evaluation and Enhancement via Probabilistic Transfer Matrices", Proceeding of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)

[4] J. von Neumann. "Probabilistic logic and the Synthesis of Reliable Organisms from Unreliable Components." Automata Studies. Princeton University Press, 1956.

[5] Jie Han and Pieter Jonker, "A System Architecture Solution for Unreliable Nanoelectronic Devices", IEEE Transactions on Nanotechnology, VOL. 1, NO. 4, DEC, 2002

[6] Yan Qi, Jianbo Gao, and José A. B. Fortes, "Markov Chains and Probabilistic Computation—A General Framework for Multiplexed Nanoelectronic Systems", IEEE Transactions on Nanotechnology, VOL. 4, NO. 2, MAR, 2005

[7] Judea Pearl, "Fusion, Propagation and Structuring in Belief Networks", Artificial Intelligence, 1986,29: 241～288

[8] Yair Weiss, "Belief Propagation and Revision in Networks with Loops", Nov 1997

[9] Cecil Huang, "Inference in belief networks-A procedural guide", International Journal of Approximate Reasoning 1994

[10] Wen-Sheng Chiang, Wen-Ta Lee, Chia-Chun Tsai, "Chip Implementation for Improved Booth Multiplier", Journal of National Taipei University of Technology, Vol. 35-1, March 2002