fQuantum : A Quantum Computing Fault Simulator

Suresh kumar Devanathan Michael L Bushnell

Abstract—We like to introduce fQuantum, a Quantum Computing Fault Simulator and new quantum computing fault model based on Hadamard, PauliX, PauliY and PauliZ gates, and the traditional *stuck-at-1* SA1 and *stuck-at-0 SA0* faults. We had close to 100% fault coverage on most circuits. The problem with lower coverage comes from function gates, which we will deal with, in future versions of this paper.

Keywords—fQuantum,Fault Simulator, Quantum Computing

I. Introduction

QUANTUM computing has been seen as the next stage of computer development. However quantum computer fabrication is prone to errors, just as must as traditional computers. Defect level testing is important to produce high quality quantum computers. We present a new quantum fault model, along side traditional stuck-at fault testing models and we did attain coverage close to 100%.

March 1, 2013

II. Quantum Fault Model

The quantum computing fault model is based on concepts similar to classical computing stuck-at models. However there are differences also. First we insert gates instead of setting a line to a particular value. Example gates include Hadamard, PauliX, PauliY and PauliZ gates.

A. Good Machine/Bad Machine Model

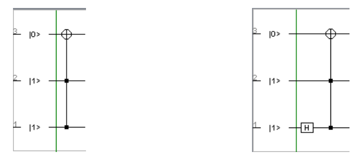
Suppose we would like to test a line in the circuit. We first do a good machine simulation. In good machine simulation, the circuit is simulated just the way it is. In bad machine simulation, we take the circuit and insert a quantum unitary transform gate, such as Hadamard, PauliX, PauliY or PauliZ and do a simulation. We then compare good and bad machine outputs, if they dont match, a fault is considered detected otherwise it is declared undetected.

B. Example Circuit

Take the following circuits, a simple Toffoli gate. Refer to figure 1 where there is a fault free circuit and one with fault inserted, i.e. a Hadamard gate. We look at the output spectra of each circuit for an input vector. If they are different a fault is considered to be detected and fQuantum does reveal fault is detectable.

III. fQuantum: Fault Simulator

fQuantum works by first generating a fault list, doing *random test generation* RTG, and for each vector in RTG, doing fault simulation and finally reporting coverage.



(a) Fault free (b) Faulty

Fig. 1: Fault Insertion Example

A. Fault List Generation

To generate a fault list, the tool goes to each line in the circuit and adds a fault of type Hadamard, PauliX, PauliZ and PauliY to the list. The input to the circuit still has a SA1 and SA0 model.

B. Vector Generation

Random vectors are generated similar to classical RTG.

C. Fault Simulation

Here's a simple fault simulation algorithm.

Algorithm 1 Bad Machine Sim algorithm

if f.gid = 0 **then**
  $InsertInputFault(f)$
else
  $InitializeRegisters()$
  for each gate g in circuit in order **do**
    if f.gid=g and f is input fault **then**
      $InsertFault(f, g)$
    end if
    $EvaluateGate(g)$
    if f.gid=g and f is output fault **then**
      $InsertFault(f, g)$
    end if
  end for
end if
$SetFinalStep()$

Algorithm 2 Good Machine Sim algorithm

$InitializeRegisters()$
for each gate g in circuit in order **do**
  $EvaluateGate(g)$
end for
$SetFinalStep()$

Algorithm 3 FSim algorithm

$GoodMachineSim()$
$xgReg = xRegister.clone()$
$ygReg = yRegister.clone()$
$FaultSimulateBadMachine(f)$
$xbReg = xRegister.clone()$
$ybReg = yRegister.clone()$
if diff(xgReg, xbReg) or diff(ygReg,ybReg) **then**
    $f.detected = Detected$
else
    $f.detected = Undetected$
end if

TABLE I: Fault Coverage

Circuit	FC
And.qc	100%
Deutsch-Josza-1.qc	100%
H-Fourier.qc	100%
Shor-437.qc	100%

IV. Results

We report sample coverage on a few test circuits using 10 RTG vectors. Results are shown in table I. We see that we generally did good.

V. Conclusion

We built a fQuantum, quantum computing fault simulator and showed it use. In the future, we will, for example, add support for function gates.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.
[2] E. Banks: "Information Processing and Transmission in Cellular Automata", MIT Ph.D. Thesis (1971)
[3] E. F. Codd: Cellular Automata, Academic Press, New York (1968).
[4] E. Fredkin, T. Toffoli: "Conservative logic", The Proceedings of the Physics of Computatio Conference in International Journal of Theoretical Physics, issue 21:3/4, 21:6/7, 21:12, pp 219-253 (1982)
[5] E. Fredkin: "An Introduction to Digital Philosophy", International Journal of Theoretical Physics, Volume 42, Number 2, pp 189-247 (2003)
[6] M. Gardner, "Wheels, Life, and Other Mathematical Amusements", W. H. Freeman and Company, New York (1983)
[7] A. Ilachinski, "Cellular Automata: A Discrete Universe", World Scientific Publishing, Singapore (2001)
[8] C. G. Langton: Self-reproduction in cellular automata, Physica, 10D, pp135144 (1984)
[9] N. Margolus: "Physics and computation", MIT Ph.D. Thesis (1987). Reprinted as Tech. Rep. MIT/LCS/TR415, MIT Lab. for Computer Science, Cambridge MA
[10] N. Margolus, "Universal cellular automata based on the collision of soft spheres", New constructions in Cellular Automata, Oxford Press (2003)
[11] K. Imai, T. Hori, K. Morita: " Self-reproduction in three-dimensional reversible cellular space", Artifical Life (MIT Press), Vol 8, Issue 2, pp 155 - 174 (2002)
[12] Two-state, Reversible, Universal CA in 3D – Miller, Fredkin
[13] J. von Neumann: The Theory of Self-Reproducing Automata. A.W. Burks (ed.), University of Illinois Press, Urbana, IL (1966).