Design of A Hardware Description Language Based Quantum Circuit Simulator

Amlan Chakrabarti¹, Susmita Sur-Kolay², Sudhindu Bikash Mandal³
¹University of Calcutta, Kolkata - 700 009, India,
Email: acakcs@caluniv.ac.in
²Indian Statistical Institute, Kolkata - 700 108, India
Email:ssk@isical.ac.in
³JIS College of Engendering, Kalyani, West Bengal, India
Email: sudhindu.mandal@gmail.com

Abstract -Quantum theory of computation establishes that there are problems for which quantum algorithms are far more efficient than their classical counterpart. Advances in quantum computing have initiated the design of systems based on quantum logic gates. Quantum computer hardware being primarily available in research laboratories at present, it mandates the ability to design, develop and test quantum logic operations by simulation of quantum logic circuits on classical computers. In this paper, we have defined a programming model for the design of a quantum circuit simulator, which can simulate a given quantum circuit obeying the rules of quantum gate operations. The quantum gate operations on both pure and superposed quantum states are being done with the help of effective data structures. The proposed quantum circuit simulator also incorporates the simulation of quantum circuit error under the action of certain quantum faults.

Index Terms-Quantum Circuit Simulator, QHDL, Bit-flip error, Error Probability Matrix

I. INTRODUCTION

The model of Quantum Computing stands on the understanding of quantum circuits and their application to solve computational problems. A quantum circuit is employed to process quantum bits (qubit)[1]. A qubit may be considered as the equivalent to a binary bit in a classical computer. It can be taken as a particular spin state of an electron, or a certain polarization state of a photon. The spin state of an electron may be up (↑) or (↓) down, or the polarization state of a photon may be vertical (↕) or horizontal (↔). The two quantum mechanical states are represented in standard quantum mechanics [2] by standard ket notation |0> and |1>. The real difference between the classical and quantum states are that while in the former, the states are definite whereas in quantum computing the states are superposed. For example, a quantum state is represented by superposition of two states like \( \psi = a |0> + b |1> \) where \( a \) and \( b \) are the complex amplitudes representing the probabilities of state \( |0> \) and \( |1> \), respectively satisfying the condition \( |a|^2 + |b|^2 = 1 \).

Unlike a classical computer in which a bit has exactly one value from the set \{0,1\}, a qubit represents both states simultaneously. The maximum number of possible states depends on the number of qubits i.e., \( n \) qubits can represent \( 2^n \) states. So a 2-qubit vector can simultaneously represent the states |00>, |01>, |10>, |11> and the probability of their occurrence depends on the complex amplitude value \( \psi = C_0 |00> + C_1 |01> + C_2 |10> + C_3 |11> \). Hence comes the concept of quantum register of \( m \) qubits holding \( 2^m \) simultaneous values. This also implies that if we perform an operation on the contents of a register, all possible values are operated on simultaneously, thus leading to quantum parallelism. However, in practice it is quite complex to achieve quantum parallelism [3], and is dependent on the property of quantum decoherence.[4]. It has been noted that by utilizing quantum parallelism certain problems can be solved in fewer steps than classical operators.

Our work focuses on the simulation methods needed for the logic verification of the quantum circuits, where we need to define the multi-qubit quantum gates in terms of unitary operators, which can be properly evaluated considering the number stages in the circuit and of distinct quantum gates in each stage.

Preliminary concepts of Quantum Circuits appear in section I, and a brief review of quantum circuit simulation in section II. Our QHDL based simulator is discussed in section III. The design methodology of QHDL is presented in section IV. Simulation steps are shown in section V. Experimental results are given in section VI. Concluding remarks appear in section VII.

II. REVIEW OF QUANTUM CIRCUIT SIMULATOR

A number of quantum simulators exists that vary in complexity, purpose, state representation and implementation. The study of quantum simulation began when Deutsch [2] introduced the notion of a Quantum Turing Machine (QTM). Many QTM simulators have been
implemented including the Quantum Turing Machine Simulator (QTS) developed by Hertel [5] in a Mathematica environment.

Most quantum simulators use complex numbers to represent quantum states. Other approaches have used Quantum Decision Diagram (QDD). A C++ library developed by Greve [6] uses binary states which are represented by Binary Decision Diagrams (BDD). This allows QDD to model relatively large quantum states although this feature limits QDD to representing a digital quantum computing model only as opposed to an analog model.

Quantum Bayesian Nets are another common representation of quantum states and are used by Quantum Fog [7] and Qubiter [8]. Using Bayesian Nets, quantum system can be represented graphically. Quantum computing language developed by Omer [9] was the first architecture-independent programming language for quantum computers. It is a quantum computer simulation language designed to work with any qubit-based quantum architecture. It is useful in studying quantum-computing theory but cannot capture hardware-specific phenomena.

The Parallel Quantum Simulator, developed by Obenland and Despain [10] was specially designed to examine the effect of errors during quantum computation. Most quantum computing simulators are designed to simulate a single algorithm on a single type of hardware, most commonly Shor’s quantum factoring algorithm and the algorithms needed to implement it.

QuaSi is a general purpose quantum circuit simulator written at the University of Karlsruhe [11]. The user is able to build and simulate quantum circuits in a graphical user interface.

The user is able to build and simulate quantum circuits in a graphical user interface. For ease of use we included three demo circuits in the simulator. The default entries are a rather good choice if you are not much familiar with the algorithms: Shor’s algorithm, Grover’s algorithm, The Deutsch-Josza algorithm to simulate circuits.

III  QHDL BASED SIMULATOR

In digital hardware design, a hardware description language or HDL [12] is any language from a class of computer languages for formal description of hardware circuits. For a given circuit, its design can be described by the language, and its operation can be verified by means of simulation. An HDL simulation program provides the hardware designer with the ability to model a piece of hardware before it is created physically.

In this research work, a QHDL parser has been designed which can take the input in a hardware description language format and can evaluate the operations accordingly. The HDL is capable of performing single gate level operations for desired inputs by generating the transfer matrix for the corresponding gate operations. A user can make a design entry of a given quantum circuit by means of the QHDL editor, and can provide the desired input combination as well. The output is the equivalent transfer matrix corresponding to a composition of all the gate operations of the desired quantum circuit, and the input state matrix. The key features of this simulator can be listed as:

- Gate level simulation
- A circuit level simulation
- Generation of transfer matrix
- Generation of output matrix
- Simulation of error performance

The speciality of this simulator compared to the existing ones is that it can simulate the error performance of a given quantum circuit according to the quantum circuit error model as described in the Section IV B below.

IV  DESIGN METHODOLOGY OF QHDL

In this section, we discuss design and basic features of the proposed simulator.

A  The Programming Model

The simulation involves (i) parsing of the HDL text description input to retrieve the gates from the syntax and also the input, (ii) formation of the input matrix from the input string, (iii) calculation of the tensor product to produce an equivalent unitary matrix for each level, and then the vector product of all these matrices to generate the circuit matrix, and finally (iv) multiplication of the circuit matrix and the input matrix to report the output matrix.

As a quantum circuit has a number of qubit lines and circuit levels, the level at which the specific gate is placed also the qubit lines on which the gate works need to be specified in the QHDL entry.

The syntax rules for the simulator are enumerated below:

1. For the gates in the library,
   - NOT: qnot(level number, qubit line number)
   - HADAMARD: qhadamard(level number, qubit line number)
   - CNOT: qcnot(level number, control qubit line number, target qubit line number)
   - SWAP: qswap(level number, qubit line number1, qubit line number2)
   - TOFFOLI: qtoffoli(level number, control1 qubit line number, control2 qubit line number, target qubit line number)
   - Controlled-SWAP(Fredkin): qoswap(level number, qubit line number1, qubit line number2, qubit line number3)

2. Gates at any particular level are delimited by “;”
3. Different levels are delimited by “;”
4. All gates in a level are to be given consecutively before the delimiter sign for that level.
5. A qubit line without a gate is represented by ( ).
Figure 1: Initial Circuit

An example of a quantum circuit is shown in Figure 1. It has 4 qubit lines ($q_1$ to $q_4$) and 3 levels ($L_1$ to $L_3$). For our simulator, it is represented by the following syntax:

\[
\text{qnot}(1,1) ; () ; () ; \text{qcnot}(2,2,3) ; () ; \text{qhadamard}(2,2) ; \text{qswap}(3,3,4) :
\]

The matrix for a level is formed by putting identity matrices at all the empty positions as shown in Figure 2 and computing the tensor product of the matrices for all the gates at that level. Having determined these matrices for all the levels, multiplying all of them in reverse order results in the matrix for the entire circuit. A single qubit (either 0 or 1) is to be entered for each input line. A $2^n \times 1$ input matrix, $n$ being the number of qubit lines, has a ‘1’ at the $i$th row where $i$ is the binary equivalent of the input combination, and rest of the entries are 0. The circuit matrix and input matrix are multiplied in order to get the output matrix.

![Figure 2: The circuit of Figure 1 with the identity gates placed for the empty qubit lines](image)

The salient features of this simulator are user-friendly interface and definition of a hardware description language for quantum computations. For the HDL portion, JAVA was a feasible choice. The basic quantum gates in the library are directly stored as class files, so that whenever computation is required involving that particular gate, the corresponding class file needs to be called.

B. Error Simulation

Our simulator supports error model for quantum NOT, HADAMARD AND TOFFOLI gates. Under bit-flip error conditions [11], a quantum gate output may have $|0>$ instead of $|1>$ and vice versa. We define $p$ as the probability of error for a gate output and hence $1-p$ is the probability for the error free output. For a given quantum gate which have $n$ number of input and output qubits we can define a error transfer matrix based on this error model. In case there are no errors the values of $p$ in the transfer matrix is replaced by 0. The transfer matrix of each quantum gate is combined using tensor or vector product of matrices for deriving the error model for the quantum circuit as a whole. As the quantum gates are also reversible the transfer matrices are of dimension $2^n \times 2^n$ where the column indices defines the different combination of the input qubits and the rows represent output qubit values. In the following, we compute the error transfer matrices corresponding to the single qubit gates like NOT and HADAMARD gates.

![NOT](image)

For $C^k$NOT (Generalized TOFFOLI) family of gates, it is assumed that the error occurs only on the target qubit. The error transfer matrices of some of the common CNOT ($k=1$) gates are presented as below:

Error Probability matrix of TOP-CNOT (top-control) gate:

![Error Prob. matrix for TOP-CNOT](image)
Error Probability matrix of BOT-CNOT (bottom-control) gate:

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1-p</td>
<td>0</td>
<td>p</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>p</td>
<td>0</td>
<td>1-p</td>
</tr>
<tr>
<td>10</td>
<td>p</td>
<td>0</td>
<td>1-p</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1-p</td>
<td>0</td>
<td>p</td>
</tr>
</tbody>
</table>

For simulating a given quantum circuit under error conditions, we utilize its error probability matrix.

C. Complexity Analysis

If the input quantum state is of \( n \) qubits, then the size of each of the intermediate and the final state vectors are \( 2^n \) and that of each of the previous and current gate matrices are \( 2^n \times 2^n \). Therefore, the total space required by the algorithm is \( 2^n \times 2^n + 2^n \times 2^n = 2^n(1 + 2^n) = 2^{2n+1} \), i.e., space complexity increases exponentially with the input size. A matrix of size \( m \times m \) when involved in tensor product with another matrix of size \( n \times n \), the size of the resultant matrix is \( mn \times mn \). So, the total number of multiplications is \( mn \times mn \). If \( m=n \), the total number of multiplications required in case of tensor product is \( n^3 \times n^3 = n^6 \), and for the vector product it is \( n^3 \). Thus the time complexity is \( O(n^6) \).

V SIMULATION STEPS

In this section we briefly on some of the key operations of the simulator.

Figure 3 shows the flow diagram of the proposed simulator with the relevant files used in each step.

A. Input

User specifies the input in QHDL syntax.

B. Syntax Checking

Grammar checking:

i. At the end of every level, there should be a “;”, otherwise an exception is thrown.
ii. If a gate is missing at a level, an exception is thrown.
iii. If user specifies an illegal gate name, an exception is thrown.

Input Label Checking:

The input in QHDL has to be given in order of the qubit lines of a gate, or according to the levels of the circuit.

C. Parser

i. Gates are recognized from the given QHDL input by delimiting every gate by “;”.
ii. For each gate name, the pattern is matched with those in Circuit.java to obtain the predefined matrix.

D. Circuit Processor

i. After getting sequence of the matrices for the levels, calculation is done by multiplying from right to left to produce the transfer matrix for the circuit.
ii. The user specified input stream is converted to an input matrix by using tensor product.

E. Output

After getting the transfer matrix and the input matrix it has been multiplied to generate the final output matrix.

VI RESULT AND ANALYSIS

Figure 4 shows the simulation window of our simulator for an example quantum circuit. The window has two sub-windows one in the left and one in the right, in the left sub-window we give the circuit definition and in the right sub-window we provide the input value of the qubits. After the simulation we obtain the transfer matrix of the circuit in the left sub-window and the output matrix in the right sub-window.
VII. Conclusion

The QHDL based simulator as discussed in this paper is a great help for quantum circuit design engineers to perform the logic verification of the circuit. The design entry of the circuit in form of HDL constructs shows a promise in further development of this tool like the commonly available EDA tools available for classical VLSI circuits. This simulator also incorporates the bit-flip error model for the quantum gates which is also a new feature considering the existing simulators in this domain. Our future work will be to incorporate some of the quantum circuit optimization rules so that we can have a simulation as well as a synthesis platform for quantum circuits.

REFERENCES


Table I

<table>
<thead>
<tr>
<th>Gate</th>
<th># Qubits</th>
<th>Simulation time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>CNOT</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>HADAMARD</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>SWAP</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>TOFFOLI</td>
<td>3</td>
<td>47</td>
</tr>
</tbody>
</table>

Table II

<table>
<thead>
<tr>
<th>Circuit</th>
<th># Qubits</th>
<th># gates</th>
<th>Simulation time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rd32</td>
<td>4</td>
<td>6</td>
<td>172</td>
</tr>
<tr>
<td>4 mod 5</td>
<td>5</td>
<td>8</td>
<td>190</td>
</tr>
<tr>
<td>5 mod 5</td>
<td>6</td>
<td>14</td>
<td>199</td>
</tr>
<tr>
<td>2 of 5</td>
<td>6</td>
<td>15</td>
<td>201</td>
</tr>
<tr>
<td>rd53</td>
<td>7</td>
<td>20</td>
<td>210</td>
</tr>
</tbody>
</table>