IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 3, July 2023

Quantum Computing Simulation

Dr. Manju S Nair, Jobin Johnson, Kiran Babu, Alex Rolands
Department of Computer Science and Engineering
College of Engineering, Adoor, Kerala, India
manjusnair@cea@ac.in, jobinrjohnson@gmail.com, kiranbabu716@gmail.com, alex.rolands123@gmail.com

Abstract: *Quantum computing is one of the emerging technologies with its immense capability, allowing us to solve computationally challenging problems more efficiently. Gaining a real quantum computer for running and testing algorithms is very costly. In order for developers and researchers to build and test algorithms, they can use simulators that can model the behaviour of a real quantum computer inside a classical computer. Simulating a quantum system on a classical machine is complex and requires a lot of processing power. This paper introduces some basic algorithms and methodologies we used to simulate an ideal multi-qubit simulator in classical terms*

Keywords: Qubit, state vector, vector space, basis states, linear transformation

I. INTRODUCTION

In recent decades, computers have advanced a lot, but there are problems beyond the capability of the current computational methods. Therefore, we need to switch to a new compu- tational paradigm capable of delivering enough computational power to solve modern day problems. Quantum computing is a newer method of computing capable of delivering exponential computational speedup. Quantum computing had gained pop- ularity in the late 20^{th} century (1994) when Peter Shor proved that a quantum system can prime factorize large numbers in polynomial time complexity. Later Lov Grover came up with an algorithm to search through an n-dimensional unstructured database in $O(\sqrt{n})$ complexity which is a way less than $O(n)$ for a classical computer.

Simulating a quantum computer on a classical computer requires exponential memory and processing power. In this paper, we are showing different approaches and algorithms we used to simulate an ideal, multi qubit, universal gate quantum computer on a classical computer.

II. QUANTUM COMPUTING CONCEPTS

2.1 Qubit

The basic unit of quantum information is called Qubit ie, Quantum bit which is a quantum alternative to the classical bit. Unlike the classical bit a Qubit can exist in a coherent superposition of both zero and one, this enables quantum computers to manipulate enormous combinations of states at once.

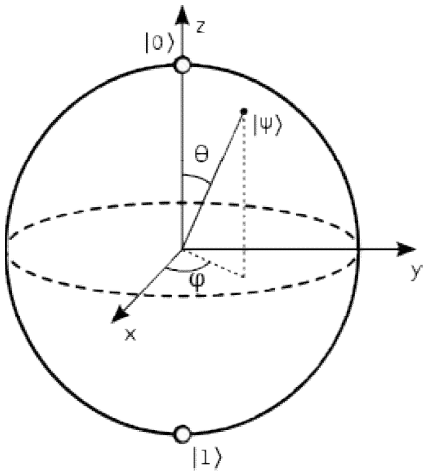

Fig. 1. Representation of a qubit using bloch sphere

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 3, July 2023

If we consider a single qubit system we can represent it int ket notation as

$$|\psi| = \alpha\,|0| + \beta\,|1|$$

where α and β are the probabilistic amplitudes of the basis states 0 *and* 1 which are complex in nature and also $\alpha^2 + \beta^2 = 1$ which is called the normalization condition. And a qubit is graphically represented by using a Bloch sphere, shown in Fig.1.

2.2 Gates

Like classical computers, there are quantum gates that operate on these qubits to change the probabilistic amplitudes of the basis states. Each quantum gate corresponds to a linear transformation. Quantum gates are reversible ie, a quantum gate is an inverse of itself. Gates can be divided into two categories, single qubit(Pauli X, Pauli Y, Hardamard, etc) and multi-qubit gates(CNOT, swap, etc) according to the number of qubits it operates on.

2.3 Measurements

At the end of the computation, we measure the qubits to obtain the results. When we measure system, the state vector will collapse on to any of the basis states based on the probabilistic amplitudes of each basis state in the state vector. After the measurement, the qubit loses its quantum properties and collapses on to a basis state that is classical.

III. ARCHITECTURE

To get maximum compatibility, we can use a client-server model to develop the system. The client machines will send the program to the server, the server will execute it and return the results back to the client. The client will present the result back to the user. Client and server are independent of each other and can be developed with different programming languages and communicate with each other via a network (if they are on different machines).

3.1 Front-end or the Client

The client machine runs a software comprising a user interface along with the functionalities of interacting with the server. This is the part where the user interacts to the system directly. It will contain an editor to input the code along with a button to execute the program. On clicking the button in the user interface, the client will send the program to the server. The server will execute the program and return the output. Front-end will parse the output and show it to the user. The client machines are independent machines they can run programs written in any languages, but they need to comply with the interfaces provided by the server.

3.2 Back-end

This will accept the request from the client machines and then validate the request, execute the program and return the response to the client. The server machine should be powerful enough to perform huge tasks. The server machine comprises two independent programs.

- *Server:* When the client makes a request, the server will check for the authorization of the client request. If the request is authorized the server will read the code from the request and validate it. if the validation complete successfully, the server will start the Quantum Simulator as a separate process and the code will be supplied to the simulator as a command line argument or a file. This can be an HTTP server or a web socket. This part will handle the authorization of the client; it also provides the interfaces to the client to communicate to the server. This acts as a wrapper to the simulator.
- *The Quantum Simulator:* This is part of the system where the real simulation happens. The quantum simulator keeps track of a state variable. This variable is reset in every execution of a quantum program. On executing each statement, the state variable gets updated. The simulator accepts the code and splits the code into statements. It will iterate over all the statements and use a deterministic finite state automata to validate the statements. After the validation if the automata is in a final state, the statement is said to be accepted. If a statement is accepted the simulator will identify the gate and extract position of the qubit on to which the gate has to be applied on to and then update the state variable by applying the specified gate on the qubits. This part of the system is explained in subsequent sections.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 3, July 2023

IV. SIMULATION

Simulation of a quantum system requires an exponential space and time complexity. Simulation of a quantum computer involves implementation of qbuits, state vector representing a system of qubits, implementation of gates and measurement.

4.1 Representation of Qubits

Classically a qubit can be represented as a two-dimensional complex vector. Similarly, an n-qubit system can be represented by a 2^n dimensional complex vector. So from the above statement, we can reach an inference that the size of the state vector increases exponentially with the number of qubits.

1 Qubit

$$|\psi| = \alpha\,|0| + \beta\,|1|\ \ State\ Vector,\ V = [\alpha, \beta]$$

2 Qubit

$$|\psi| = \alpha\,|00| + \beta\,|01| + \gamma\,|10| + \delta\,|11|\ \ State\ Vector,\ V = [\alpha, \beta, \gamma, \delta]$$

In the above example the state vector representing two qubit system is a Kronecker product or tensor product of state vector representing of two single qubit systems.

4.2 Implementation of Quantum Gates

Every quantum gate corresponds to a transformation. Apply- ing a gate to the state vector can be done by simply multiplying the gate matrix with the state vector.

$$Q = M \times Q \qquad (1)$$

$$q1 = a \times b \times q1 \qquad (2)$$

$$q2 = c \times d \times q2 \qquad (3)$$

Where Q is the new state, M is the transformation matrix corresponding to the gate and Q is the initial state vector.

The matrices for single qubit gates are of order 2×2 so we can only apply it on a state vector of size 2. Similar in the case of two qubit gates, CNOT and SWAP matrix in its native form can only be applied on to a state vector of size 2 i.e., a system of 2 Qubits. If have only one qubit, there won't be a problem. When dealing with a system of qubits we need to expand the above matrix to generate a new matrix which is capable of applying the gate on specific qubits in a multi qubit state vector.

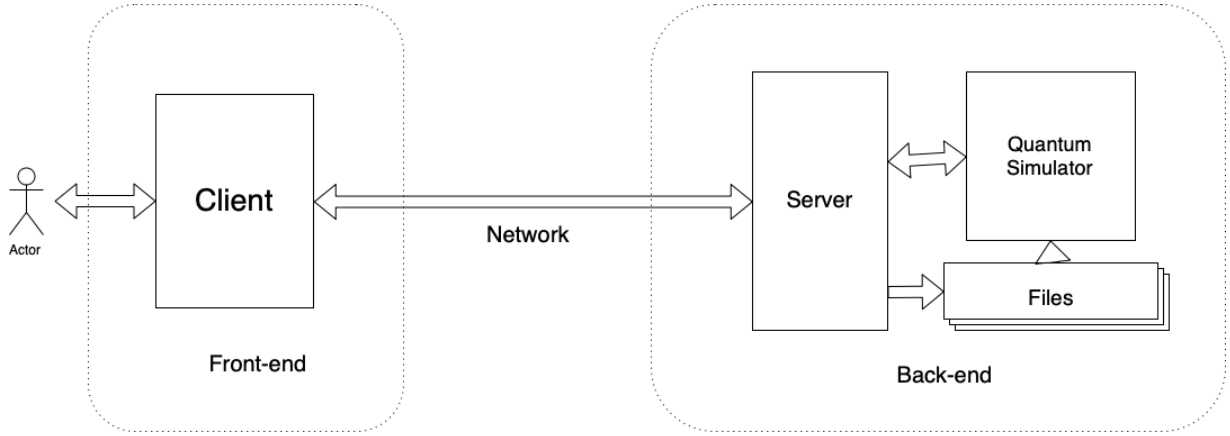

Fig. 2. Architecture of the proposed simulator

TABLE I: Some commonly used Quantum gates and their transformation matrices

Gate	*Matrix*	*Description*
Pauli X	$\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}$	π rotation around the X Axis
Pauli Y	$\begin{matrix} 1 & 0 \\ 0 & -i \end{matrix}$	π rotation around the Y Axis
Pauli Z	$\begin{matrix} 1 & 0 \end{matrix}$	π rotation around the Z Axis

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 3, July 2023

	$0-1$	
Hardamard	$\frac{\sqrt{1}}{2}\begin{matrix}1 & 1\\ 1 & 1\end{matrix}$	Create superposition
T	$\begin{matrix}1 & 0\\ 0 & e^{\pi/4}\end{matrix}$	$\pi/4$ rotation around Z Axis
CNOT	$\begin{matrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{matrix}$	Controlled NOT. It takes a control bit ad target bit. It flips the target bit if control bit is set
SWAP	$\begin{matrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{matrix}$	Swaps two qubits

4.3 Applying Single Qubit gates

When we need to apply a gate on a specific qubit inside a system of qubits. If we have n qubits our state vector will be a 2^n dimensional vector. So we need to expand a given 2×2 transformation matrix to a $2^n \times 2^n$ matrix. This can be done using the below algorithm.

In the above algorithm gate matrix is a 2×2 matrix which operates on a single qubit (ie, state vector of size 2). state_vector represents the current state of the system of qubits which is of size 2^n and n is the number of qubits and i is the position of the qubit on which the gate has to be applied. The variable identity refers to a 2×2 identity matrix. Here we assume an operator $\otimes$ which computes the Tensor product of two matrices a and b also if a is null it will return b.

Input: *state_vector*, *gate_matrix*, i position of the qubit, n as the number of qubits
Output: *state_vector* after applying *gate_matrix* on i^{th} position $new\_matrix = null$ $counter = 0$
while $counter < n^n$ **do if** $counter == i$ **then**

$mat = gate\_matrix$
else
$mat = identity$
end
$new\_matrix = new\_matrix \otimes mat$
$counter = counter + 1$
end
$state\_vector = state\_vector\ X\ new\_matrix$
return *state_vector*

Algorithm 1: Algorithm for applying single qubit gate on a specified position

4.4 Implementing multi Qubit Gates

The above algorithm cannot be used with multi qubit gate as it requires 2 qubits as input. There are different approaches for performing these operations. If we need to apply gates on any qubits at arbitrary positions, we can do this by

- By iterating through all possible states and set the matrix according to the purpose of the gate
- By matrix expansion, similar to above described method.

Both the methods requires equal time complexity. But the first is known for its simplicity

Implementation of SWAP gate: Here we consider the second method to generate the matrix. The algorithm takes in state_vector which represents the current state of the system of qubits having size 2^n, n represents the number of qubits and i and j which are any arbitrary positions of the qubit on which the gate has to be applied.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 3, July 2023

In the algorithm first we initialize an empty gate_matrix of size $2^n \times 2^n$, then we iterate over all possible states

Input: *state_vector*, *a* position of control qubit, *b* position for target qubit
Output: *state_vector* after applying CNOT on *b* with *a* as control qubit *gate_matrix* = [][] *counter* = 0
while *counter* < 2^n **do**
    a = *counter*&(1 << (*n* - *a* - 1)) == 0
    b = *counter*&(1 << (*n* - *b* - 1)) == 0
    ctr2 := 0
    row = *null*
while *ctr2* < *n* **do**
    m := *counter*&(1 << (*n* - *ctr2* - 1)) == 0
    if *ctr2* == *counter* **then**
        m = *a*
    else if *ctr2* == *j* **then**
        m = *b*
    if *m* == 1 **then**
        mat = [0, 1]
    else
        mat = [0, 1]
    end
        row = *row* ⊗ *mat*
        ctr2 = *ctr2* + *1*
    end
        gate_matrix[*counter*] = *row counter* = *counter* + 1
end
    state_vector = *state_vector X gate_matrix*
return *state_vector*
Algorithm 2: Algorithm for CNOT gate implementation

Here the variable counter holds all the possible states. For example a 2 qubit system can have states 00, 01, 10 and 11 which corresponds to 0, 1, 2and 3. In the second loop we iterate over all the bits in the states. Here the operator & implies the bitwise and operator, << implies the left shift operator and implies the tensor product operator. We find the values of qubits corresponding to i and j each possible states and store it in a and b respectively.

Then we iterate over each bit positions in every states if the bit positions match with i or j we exchange the bits. Then we create a state matrix and perform tensor product with the row matrix. We continue computing row matrix for each state and store it the gate_matrix array. At the end we multiply the gate_matrix with state_vector to obtain the new state vector.

In the same manner we can implement the controlled swap or FREDKIN gate. In that case we need to use a control qubit, let the position of it be *k*. For each state check if the bit at k^{th} position is 1 and then swap the values.

Implementation of CNOT gate: CNOT or Controlled NOT can also be implemented in the above manner. Use one qubit as a control qubit(say i^{th} qubit) and other one as a target Qubit(say j^{th}). CNOT gate will flip the target qubit if the control qubit is set to 1 otherwise it will leave the control qubit unchanged. In the above algorithm instead of swapping, flip the state of the bit if the bit at i^{th} position is set.

Input: *state_vector*, *a* position of first qubit, *b* position for second qubit
Output: *state_vector* after swapping on *a* and *b* *gate_matrix* = [][]
counter = 0

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 3, July 2023

while $counter < 2^n$ **do**
    $a = counter\&(1 << (n - a - 1)) == 0$
    $b = counter\&(1 << (n - b - 1)) == 0$
    $ctr2 := 0$
    $row = null$
while $ctr2 < n$ **do**
    $m := counter\&(1 << (n - ctr2 - 1)) == 0$ - -
    if $a == 1$ **then**
        $m = !m$
    if $m == 1$ **then**
        $mat = [0, 1]$
    else
        $mat = [0, 1]$
    end
        $row = row \otimes mat$
        $ctr2 = ctr2 + 1$
    end
        $gatematrix[counter] = row\ counter = counter + 1$
    end
        $state\_vector = state\_vector\ X\ gate\_matrix$
    return $state\_vector$

Algorithm 3: Algorithm for SWAP gate implementation

Measurement: This is the final step in every quantum algorithm. Here we need to select a qubit based on the probabilistic amplitudes and return its binary value. Here we create a circle with the probabilities as its arcs. Then choose a random number r such that $0 <= r <= 1$ and find the arc in which the random we have chosen is present. Then return the binary value corresponding to that state.

Input: $state\_vector$
Output: $state$ after measuring the system
    $r = randomnumber\ circle = 0$
    $counter = 0$
    for each $state \in state\_vector$ **do**
        $amplitude = state^2$
        $circle = circle + amplitude$
    if $circle = r$ **then return** $counter$
        $counter = counter + 1$
    end
    return $counter$ 1 –

Algorithm 4: Algorithm for measurement

V. PERFORMANCE ANALYSIS

Performance is one of the main aspects of any simulation. In this section, we analyze the time and space complexities of the algorithms we used. The basic concept is, a simulation of a quantum system in classical terms cannot be done without compromising the exponential amount of processing power. Here, the order of time and space complexities changes exponentially with the change in the number of qubits

5.1 Space Complexity

The time and space complexity of the system varies expo- nentially with the number of qubits. For an n qubit system, the length of the state vector should be 2^n. Since the amplitudes of the basis states are complex in nature, we need two

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 3, July 2023

variables to represent a state one for the real part and other for the complex part and these numbers are double precision floating- point numbers.

Consider a single qubit system;It has 2 elements in the state vector. So the amount of memory it consumes is (assuming sizeof double precision floating point number on the architec- ture is 8 bytes):

$$Size = 2^n * 2(for\ complex\ and\ real\ part) * sizeof(double) = 16\ bytes$$

Lets consider a 2 qubit system, which has 4 basis states and the amount of memory consumed for the state vector will be 64 bytes. If we continue further and increase the number of the qubits to 16 the state vector will consume around 1 MB of memory.

Similarly, if we consider the implementation of gates, each gate is a matrix of size $2^n \quad 2^n$, the advantage here is most of the gate matrices contains real elements, also most of the elements in a gate matrix is zero. Due to this nature, this can be implemented in the form of a sparse matrix that can reduce a significant amount of memory. The above-mentioned implementations of gate matrices can be a bit sophisticated but can deliver better results in terms of memory conception. If we consider a simpler implementation, having a matrix size of $2^n \quad 2^n$ and each element is a complex number. The memory requirement will be

$$Memory = 2^n \times 2^n \times 2^n \times sizeof(double) \qquad (3)$$

With a single qubit system it can occupy 64 bytes, and if we go on increasing the number of qubits to 16 the transformation matrix will occupy upto 4 GB, which is a large number. But if we use the sparse matrix implementation, we can reduce the space for the matrix to a greater extent.

5.2 Time Complexity

The numbers are almost similar for the time complexity. The main part of the time is spent expanding the matrix and multiplying the gate matrix with the state vector. If we consider an n qubit system, expanding it requires a time complexity of $O(2^n)$, similarly for applying the gate matrix on to a vector for an n qubit system requires a time complexity of $O(2^{2n})$. But the real world performance of the system depends on the complexity of the gate, number of qubits the gate applies on to, the level of entanglement of the system, the presence of complex elements on the transformation matrix, the CPU architecture, the type of memory management used by the operating system, etc.

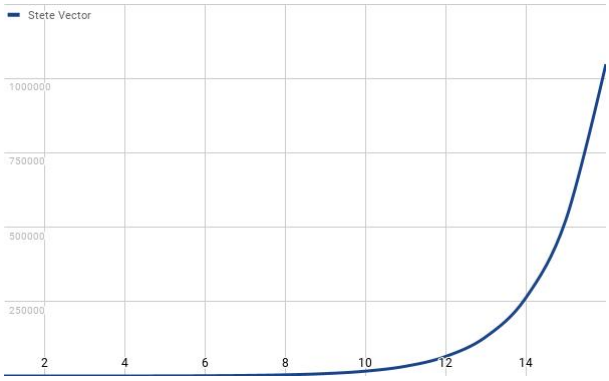

Fig. 3. Size of the state vector as a function of the number of qubits contained in the system

The speed of the system can be vastly increased using parallel processing. The nature of the system is well suitable for GPGPU (General purpose Graphics processing unit) type computation as this involves only matrix and vector processing.

VI. CONCLUSION

By using the above approaches, we were successfully able to simulate a quantum computer on a classical computer. It can be seen that the computational complexity of the system scales with the number of qubits. It also depends on the complexity of the gates used and the level of entanglement of the system. The real world performance of the system also depends on the hardware architecture, the language and libraries used and the use of the programming constructs

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 3, July 2023

REFERENCES

[1] Valentin Pavlov Efficient Quantum Computing Simulation, Bulgarian Academy of Sciences Institute for Parallel Processing , 2006

[2] Yves Vandriessche, Ellie DHondt, Tom Van Cutsem and Theo DHondt, Software Languages Lab, Vrije Universiteit Brussel,A highly-parallel formulation of quantum computing simulation through fine-grained dataflow

[3] Kamatchi Priya, R.Latha, T.Vijila, E.Srividhya, Vikash Kumar Jha, Department of Computer Science and Engineering Aarupadai Veedu Institute of Technology, Literature Survey on Quantum Computing

[4] Kevin M. Obenland and Andrew J. Kerman , MIT Lincoln Laboratory Realistic Simulation of Error in Quantum Computing Circuits