Quantum Computing Simulator
Term Project

Mankrit Singh Kalra

1st September 2023

Abstract

This C program offers a window into the elusive realm of quantum mechanics, turning the abstract concepts of superposition and entanglement into tangible simulations. At its core, the software simulates fundamental quantum operations on a system of n qubits. These operations are depicted using a state vector, where each element mirrors a potential qubit state and is characterized as a complex number signifying its amplitude. The software includes essential quantum gates like Hadamard, NOT, Z, and Phase Shift, alongside more intricate operations such as the Quantum Fourier Transform (QFT). The QFT is crucial in various quantum algorithms, transforming qubit states from the computational basis to the Fourier basis. The simulation provides an initialization procedure, setting the quantum state to predefined values, and displays the state before and after applying the QFT. While not a replacement for genuine quantum hardware, this program is an invaluable educational resource, offering insights into quantum algorithms' behavior and aiding in their understanding and testing.

1 Introduction

This project aims to demonstrate the behaviour of several quantum gates and the Quantum fourier transform.The code has been written in C. There are three files, *qc.c*,qc_sq.c and applygateqc.c. I initially started with qc_sq.c which deals with a single qubit and different quantum gates whose workings I will be explaining in this document. *qc.c* deals with an array of qubits represented using structs. Finally, The applygateqc.c makes use of State Vector Representation which allows for a more holistic representation of multi qubit states. Both qc.c and applygateqc.c simulate a Quantum fourier transform.

2 Quantum Gates

2.1 NOT gate/X gate

The NOT (or X) gate flips the values of the qubits zero and one states.
The matrix used to apply the NOT gate in quantum computing is called the Pauli-X matrix. It is a 2×2 matrix defined as:

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

When applied to a qubit in the computational basis, it switches the amplitudes of the $|0\rangle$ and $|1\rangle$ states. In other words, if a qubit is initially in the $|0\rangle$ state, applying the Pauli-X matrix will put it in the $|1\rangle$ state, and vice versa. The Pauli-X matrix is sometimes simply referred to as the X gate.

2.2 Hadamard Gate

The Hadamard or Walsh-Hadamard gate, named after Jacques Hadamard and Joseph L. Walsh, acts on a single qubit. it creates an equal superposition state if given a computational basis state. The two states

$(—0\rangle + |1\rangle)/\sqrt{2}$ and

$(—0\rangle - |1\rangle)/\sqrt{2}$

The Hadamard gate performs a rotation of π about the axis $(\hat{x} + \hat{z})/\sqrt{2}$

It is represented by the Hadamard matrix:

$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$

2.3 CNOT Gate

The CNOT gate has a control qubit and a target qubit. If the control qubit is in the $|1\rangle$ state, it flips the value of the target qubit and otherwise leaves it unchanged.

2.4 Z Gate

The Pauli Z gate rotates a qubit around the Z axis by π radians. Pauli Z leaves the basis state ($|0\rangle$ unchanged and maps $|1\rangle$ to $-|1\rangle$. It is also known as a phase flip. It is represented by the matrix: $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

2.5 Phase shift gates

The phase shift is a family of single-qubit gates that map the basis states

$—0\rangle \mapsto |0\rangle$ and

$—1\rangle \mapsto e^{i\varphi}|1\rangle$. The probability of measuring a$|0\rangle$ or $|1\rangle$ is unchanged after applying this gate, however it modifies the phase of the quantum state.

3 Quantum Fourier Transform

The QFT is a quantum analogue of the classical discrete Fourier transform (DFT), which is a mathematical operation that decomposes a finite sequence of complex numbers into a sum of sine and cosine waves of different frequencies. The QFT similarly decomposes a quantum state, which is represented as a superposition of states, into a sum of states with different frequencies.
In quantum computing, the input to the QFT is a quantum state —x⟩ represented by a sequence of n qubits. The QFT operates on this quantum state and produces a new quantum state —y⟩, which is the Fourier transform of —x⟩. The QFT can be expressed mathematically as follows:
QFT(—x⟩) = $\frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} e^{2\pi i x y/2^n} |y\rangle$

4 Explanation of the Code and functions used

4.1 qc_sq.c

This is a C program that defines several functions to simulate quantum gates and applies those gates to qubits. It also defines a main function to test the implemented quantum gates.

The program defines the R constant, which is used in the Hadamard and Z gates.A qubit struct is defined to store the values of each qubit.

Next, the program defines several functions to simulate quantum gates:

printQubit(struct qubit q) : This function prints the value of a qubit in the standard Dirac notation.

NOT(struct qubit* q) : This function applies the NOT (or X) gate to a qubit.

H(struct qubit* q) : This function applies the Hadamard gate to a qubit.

CNOT(struct qubit* control, struct qubit* target) : This function applies the CNOT gate between a control qubit and a target qubit.

Z(struct qubit* q) : This function applies the Z gate to a qubit, which is a rotation around the Z-axis by πradians.

phaseShift(struct qubit* q, int k, int n) : This function applies the phase shift gate to a qubit. The gate rotates the qubit state around the Z-axis by an angle determined by k and n.

controlPhaseShift(struct qubit* q1, struct qubit* q2, int k, int n) : This function applies the controlled phase shift gate to two qubits. The gate rotates the target qubit around the Z-axis by an angle determined by k and n if the control qubit is in the —1⟩ state.

After defining the quantum gates, the program defines the main() function, which creates three qubits and applies some of the implemented gates to them.

4.2 qc.c

This is a C program that implements the Quantum Fourier Transform (QFT) algorithm on an array of qubits.

Similar to the previous program, this code defines a qubit struct, which has two float fields representing the probabilities of measuring the qubit in the —0¿ and —1¿ states, respectively. The code also defines several functions for applying quantum gates to qubits and a function for applying the QFT to an array of qubits.

The code includes the necessary header files and defines two global variables. R is a constant representing the square root of 1/2, and n is an integer variable representing the number of qubits.
an array of qubit structs named qa is created, with three elements initialized to specific values. This represents the initial state of the qubits.
Next, the program defines some functions to manipulate qubits, such as printQubit, X, NOT, H, CNOT, Z, phaseShift, controlPhaseShift, and QFT.
The printQubit function is a simple utility function that takes a qubit as an argument and prints its values in the standard Dirac notation.

The X function applies the X gate (also known as the NOT gate) to a qubit.

The NOT function is similar to the X function, but instead of returning a new qubit, it modifies the qubit passed as an argument.

The H function applies the Hadamard gate to a qubit.

The CNOT function applies the controlled NOT gate to two qubits, where the first qubit is the control qubit and the second qubit is the target qubit.

The Z function applies the Z gate to a qubit.

The phaseShift function applies a phase shift gate to a qubit with a given phase shift value.

The controlPhaseShift function applies a controlled phase shift gate to two qubits, where the first qubit is the control qubit and the second qubit is the target qubit.

The QFT function implements the QFT algorithm on an array of qubits. The function first applies the Hadamard gate to each qubit, then applies the controlled phase shift gates to each subsequent qubit, and finally reverses the order of the qubits.

In the main function, the program calls the QFT function on the qa array of qubits, and then prints the values of the qubits using the printQubit function.

The output of the quantum Fourier transform (QFT) applied to a set of qubits represents the probability amplitudes of measuring the qubits in a specific state after performing the transform. Specifically, if the input to the QFT is a set of n qubits, then the output is a set of probability amplitudes corresponding to each of the 2^n possible states of the qubits.

Overall, this program demonstrates the use of the QFT algorithm to manipulate qubits in quantum computing.

4.3 applygateqc.c

4.3.1 Global Variables and Constants:

const int n: The number of qubits in the system.

const int N: The size of the state vector, which is 2^n. This indicates that there are 2^n possible states for n qubits.

double complex state[N]: The state vector that represents the quantum system. Each element of the state vector corresponds to a possible state of the qubits, and its value is a complex number that indicates the amplitude of that state.

4.3.2 Utility Functions:

normalize(): Ensures that the quantum state vector has a magnitude (norm) of 1. This normalization is essential as quantum states must always be normalized.

initialize(): function sets our system to a predefined quantum state. This specific initialization results in a superposition of states .$|001\rangle, |010\rangle, |110\rangle$, and $|111\rangle$.

printState(): Outputs the current state of the quantum system, presenting the amplitude of each state and its corresponding binary representation.

applyGate(): A generic function that applies a 1-qubit gate, represented as a 2x2 matrix, to a specific qubit of the state vector.

4.3.3 Execution Flow:

In the main() function, the program initializes a state vector, prints its values, applies the Quantum Fourier Transform, and then prints the final state.

4.3.4 Notes

Quantum gates, when applied to a normalized state, should always produce a normalized state. Hence, the frequent use of normalize() is mostly for safety and may not be required after every gate application.

4.3.5 Conclusion

This program provides a basic simulation of quantum operations, allowing one to visualize the impact of quantum gates and transformations on a quantum state. While this simulation is not a substitute for real quantum hardware, it serves as an educational tool for understanding and testing quantum algorithms.