Simulating a Quantum Computer

Cory Schillaci

July 26, 2012

Now that you have learned a little bit about quantum mechanics and quantum computation, it's time to put it all into practice. You will work with a software program called jQuantum that implements the basic logic gates and operations of quantum computation. After learning the basics of how to use jQuantum, you will implement some quantum algorithms so you can see them in action.

1 Learning jQuantum

The first step is to get familiar with the jQuantum program. Depending on your learning style, you may want to start by just playing around with the program. You can also read the second section of the user manual, but it's pretty sparse. The teaching assistants will be on hand to help if you have any questions.

When you feel like you know where things are, go through the following sections to make sure.

1.1 Initializing a state

There are two steps to initializing a state. First, you must decide on the number of qubits you would like to have in each register. This can be done by clicking on the bottom labelled $|\psi_0\rangle$ in the upper left corner of the screen. In this section, please initialize three qubits in the x-register and two qubits in the y-register.

Once you have selected the the register sizes, there are two ways to set the initial value of each qubit. The first option showed up when you were setting up the registers, and allowed you to load an initial state into the x-register corresponding to the binary number representing that state (more on this in section 1.2). A more direct method is to click the leftmost button under "Circuit Design," the one with a $|1\rangle$ and a $|0\rangle$ stacked on top of one another. Clicking on each qubit toggles its initial value between 0 and 1. For the purposes of this tutorial, set up the x-register to be in the state $|001\rangle$ and the y-register as $|11\rangle$.

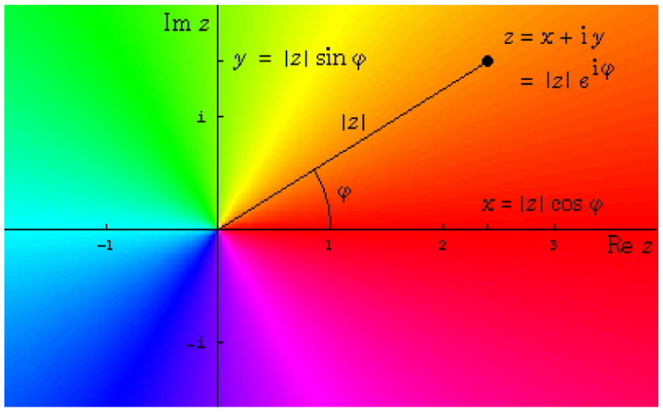

Figure 1: The color map of the complex plane. The hue depends uniquely on the phase ϕ. (The hue of the origin $z = 0$ remains undetermined, we define it here as black). For instance, a positive real number is red, a negative one is light blue

1.2 Understanding the register

You will notice that the bottom of the screen contains two displays labelled as the x-register and y-register. These are filled with boxes, all of which are initially colored red and black. Take a minute to see if you can explain how the correspondence between the boxes and the qubit states works. Hint: holding your mouse over a box shows the state that the box represents.

Once you have an idea, or if you are stumped, ask a TA to confirm your answer or to help you figure it out.

Since initial states in jQuantum are always pure and have real coefficients, the boxes will always start as red and black. The color actually represents the phase of the state, see Figure 1 (taken from the user manual's section 2.3). In the next section we will use a T gate to change the phase of a state and watch the effect on the color of the boxes.

1.3 Building and executing a circuit

Here we will construct a very simple example circuit and see how to execute it. The row of buttons underneath "Circuit Design" at the top of the window allows you to insert operations into your circuit. Click on the T gate (make sure you remember what this gate does!) and a box will popup. Target qubit 2 in the x-register and click OK. The T gate will now appear on the circuit diagram above the registers.

Wait! I actually wanted you to apply it to qubit 1, so now you'll have to delete the gate. Use the bright red left arrow to remove this "accident," then apply the T gate to the correct qubit. Note the jQuantum only allows you to remove the most recent operation. Older mistakes unfortunately require that you delete all subsequent gates as well. To complete our extremely simple circuit, add a measurement on the x-register. The measurement operation is carried out by that big red M.

It's now time to watch the circuit in (simulated) action! The green arrows with the black lines go one step at a time, while those without jump to the beginning or the end of the circuit in a single click. Increment the circuit we have built a single step. What changed? Can you write the current state in braket notation? Increment it one more time, which performs the measurement. Can you explain exactly what this circuit has done? Hint: it should look exactly like the initial state we set up in 1.1.

Congratulations, you've now built at least one circuit and run through it with jQuantum. Now it's time for the fun stuff.

2 Algorithms in action

Now you are set to implement some real quantum algorithms and watch them work. This should help give you a feel for what is going on as these do their work. The following problems will work you through some simple problem solving and a few of the major algorithms as you become more and more comfortable with the quantum computing paradigm.

Problem 2.1 Preparing entangled states

Starting from a pure initial state with two qubits in the x-register, prepare the Bell state

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

with as few operations as possible. What happens when you measure this state? Do it a couple of times and confirm your prediction.

Problem 2.2 Quantum Teleporter

Using entangled states, it is possible for one observer to transmit their quantum state exactly to another. Let one observer, call her Alice, start with a state $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$. Assume that the coefficients satisfy the usual normalization condition. The idea is that she will share part of a Bell state with another observer, named Bob.

1. Set up the x-register with three qubits. The first two qubits belong to Alice, while the third belongs to Bob.

2. Prepare the first qubit in any state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ you wish. It's more interesting if you make both α and β non-zero.
3. Use the method you developed in Problem 2.1 to create an entangled state using the second and third qubit. After this point, Bob and Alice can go their separate ways while hanging onto their qubits for later. Bob doesn't know anything about qubit 1.
4. Create the circuit from Figure 2

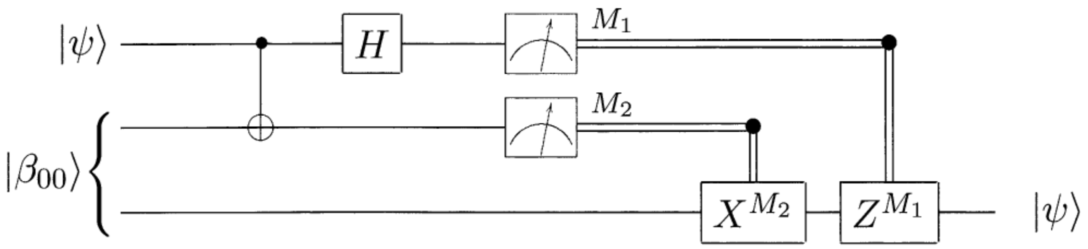

Figure 2: The quantum teleporter circuit

5. Now Alice tells Bob what she measured. If she measured qubit 1 to be M_1 and qubit 2 to be M_2, Bob first uses an X gate M_2 times and a Z gate M_1 times (where these could be zero).
6. Voila, Bob has a copy of the state stored in Alice's qubit 1. Play around with this several times and observe that it always works. Also note that Alice's state is destroyed in the process, so this algorithm doesn't duplicate the state.

Problem 2.3 Function evaluation gate

Knowing how the function evaluation gate works is crucial in order to understand the Deutsch-Josza algorithm in Problem 2.4. For the implementation in jQuantum, you must have states in both registers. I believe this is to ensure that the state factorizes, there is no deep reason that the function evaluation requires states in different "registers."

Call the number of qubits in the y-register n_y. Letting the state in the x(y)-register be $|x\rangle$ ($|y\rangle$), we can write the effect of the function evaluation as

$$|x\rangle|y\rangle \to |x\rangle|(y + f(x)) \bmod 2^{n_y}\rangle$$

For example, if $n = 2$, $x = 5$, $y = 0$, and $f(x) = 2x + 3$, we would have

$$|5\rangle|0\rangle \to |5\rangle|(0 + 13) \bmod 4\rangle = |5\rangle|1\rangle$$

Choose several different register sizes and functions, trying to predict the outcome. Once you feel comfortable that you understand how this gate works, move on to the next problem.

Problem 2.4 Deutsch-Josza algorithm

I will quote Nielsen and Chuang's seminal textbook to describe the setup for this problem:

> Deutsch's algorithm is a simple case of a more general quantum algorithm, which we shall refer to as the Deutsch-Jozsa algorithm. The application, known as Deutsch's problem, may be described as the following game. Alice, in Amsterdam, selects a number x from 0 to $2^n - 1$, and mails it in a letter to Bob, in Boston. Bob calculates some function $f(x)$ and replies with the result, which is either 0 or 1. Now, Bob has promised to use a function which is of one of two kinds; either $f(x)$ is constant for all values of x, or else $f(x)$ is balanced, that is, equal to 1 for exactly half of all the possible x, and 0 for the other half. Alice's goal is to determine with certainty whether Bob has chosen a constant or a balanced function, corresponding with him as little as possible. How fast can she succeed.?

First, give some examples of constant functions and balanced functions. Give an example or two of a function that is neither.

To implement the algorithm:

1. start with an x-register of two or more qubits in the zero state and one qubit in the $|1\rangle$ state in the y-register.
2. Apply a Hadamard gate to each of the qubits in both registers.
3. Apply a function evaluation.
4. Apply the Hadamard gate to each of the x-register qubits.
5. Now insert a measurement.
6. Run the algorithm. If you measure 0 in the x-register, the function is constant. Otherwise, the function is balanced.

Try the algorithm with a couple of different functions. What happens if you use a function that is neither balanced nor constant?

Note: If you want to learn more about David Deutsch, the Deutsch of this algorithm and one of the founders of quantum computing, I have posted a relevant article from the May 2, 2011 edition of the New Yorker. The article also mentions Robert Schoelkopf's lab at Yale, where two qubit algorithms have been successfully realized.

Problem 2.5 Reversible computation is universal

The classical logic gates NAND and FANOUT are universal, meaning that all classical gates can be constructed from these two. The truth table for NAND (short for "not and") is given in Figure 3. Bits are copied using the FANOUT operation. Although these are not unitary classically, you can use the Toffoli gate to reproduce these in a reversible way. The secret is to use three qubits, storing the output in a third bit without altering the inputs.

a) Demonstrate an implementation of a NAND gate with the Toffoli gate in jQuantum.

b) Implement the FANOUT operation using a Toffoli gate in jQuantum. Note that you are copying a quantum state. Does this violate the no-cloning theorem?

This is important because Landauer's principle requires the dissipation of energy (alternately, the increase of entropy) whenever information is erased. Classically, this provides a lower bound on the energy to perform computations. Showing that a quantum computer can reversibly simulate a classical computer implies that ideal computation does not fundamentally require energy.

a	b	$\neg(a \wedge b)$
0	0	1
0	1	1
1	0	1
1	1	0

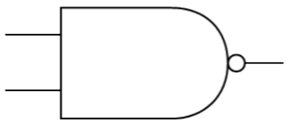

Figure 3: Truth table for the classical NAND gate, the negation of the AND gate. Symbolically, NAND is given by $\neg(a \wedge b)$.