Lecture 5: Computation using circuits

Rajat Mittal

IIT Kanpur

These notes will focus on the circuit model of computation. We will first start with a toy problem and show a quantum advantage to it (Deutsch's algorithm). This will provide motivation to look at quantum circuits.

1 Deutsch's algorithm

Let us reintroduce few basic concepts of quantum computing, that would be sufficient to understand this algorithm.

Qubits: The fundamental unit of computation is a bit on a classical computer. A bit can be in two states, either in 0 or in 1. The analog of a bit on a quantum computer is called a *qubit*. A qubit can be in the state $|0\rangle$ or $|1\rangle$ (corresponding to the classical states) or even in a superposition of these states. Specifically, the state of a qubit is

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle.$$

Where α, β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$.

Exercise 1. Why do we have a mathematical equation for a qubit. Shouldn't it be described by a physical object?

The states $|0\rangle$ and $|1\rangle$ are called the basis states and the above equation states: any linear combination of the basis states with *length* 1 is a valid state. While talking about the length of a state, we view it as a vector. Remember that this corresponds to the first postulate of quantum mechanics.

Importantly, it is not possible to figure out α, β given a physical qubit in state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. We are only allowed to *measure* the qubit. Let us take an example of a measurement. If we measure in the standard basis, with basis states $|0\rangle$ and $|1\rangle$, then with $|\alpha|^2$ probability we will observe state $|0\rangle$ and with $|\beta|^2$ probability we will get state $|1\rangle$. This provides more justification as to why the norm of a state should be 1.

Like in the case of classical computing, we use multiple qubits to store the data in a quantum computer. What are the possible states of two qubits?

The basis states should be $|0\rangle|0\rangle, |0\rangle|1\rangle, |1\rangle|0\rangle$ and $|1\rangle|1\rangle$. We will identify the state $|0\rangle|0\rangle$ with the state $|00\rangle$ and similarly for other basis states. As before, we will say that any linear combination of these states will be a valid state.

$$|\psi\rangle = \alpha_1|00\rangle + \alpha_2|01\rangle + \alpha_3|10\rangle + \alpha_4|11\rangle.$$

The length of this state should be 1, so $\sum_i |\alpha_i|^2 = 1$.

Operations: We have defined states of different parts of the quantum computer. The next step would be, what kind of operations can we perform on this computer? In other words, we are searching for the analog of NOT/AND/NAND/OR gates on a quantum computer.

For simplicity, we will start with single qubit gates, gates which act on only one qubit. We can define the quantum NOT gate which takes $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$.

Exercise 2. Is this description sufficient?

Since our state space is big, any linear combination of $|0\rangle$ and $|1\rangle$, we need to define the action on the complete state space. Using the postulates of quantum mechanics, this is done linearly. So for a quantum NOT gate, $\alpha|0\rangle + \beta|1\rangle$ goes to $\alpha|1\rangle + \beta|0\rangle$.

Note 1. This means that to specify a quantum gate, we only need to mention its action on the basis elements.

Exercise 3. What are the other bases for the state of a quantum bit?

Linearity implies that any quantum gate on a single qubit can be written as a 2×2 matrix (if there are n qubits, then $2^n \times 2^n$ matrix). Since any state has unit norm, these matrices should be *unitary.* The matrix representation of quantum NOT gate is pretty simple and it is equal to Pauli X operator.

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

There are infinite possible single qubit gates (unitary 2×2 matrix) as compared to finitely many classical gates. Can we realize each of them on a quantum computer? Should we build a quantum computer which has all these gates?

Thankfully, It turns out that there are elementary gates which can be composed to perform any possible quantum gate approximately (even for the case of multiple qubit gates). Whether this can be done efficiently, that is a question of great interest.

Few other interesting gates we saw were *Hadamard* and *controlled NOT (CNOT)* gate.

Exercise 4. What do you think CCNOT gate should be? Write the matrix representation of CCNOT gate.

Subroutine in a quantum computer:

Exercise 5. What should be the equivalent of a subroutine (say on input x we get $f(x)$ classically) in a quantum computer?

Linearity in the circuit allows us to apply an operation/function on many inputs simultaneously. Notice that we said apply and not compute. This distinction will be clear after the explanation below, but you should be very careful in drawing conclusions from the next paragraph.

Suppose there is a circuit which evaluates $f(x)$ given an input x. The theory of quantum mechanics forces any operation to be reversible. So, a gate which outputs $f(x)$ on the input x is not a valid quantum gate (why?).

The same objective of computing $f(x)$ can be achieved in multiple ways, one of the possible implementations is given in the diagram below (Fig. 1). To compute the function value on a bit b, we set the control qubit to 0 and the data qubit to b (take a look at the diagram). This is called an *oracle/black-box for f.*

Exercise 6. What will happen if we feed the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ as the data qubit and $|0\rangle$ as the control qubit to the gate given above?

The output, by linearity, should be,

$$\frac{1}{\sqrt{2}}(|0, f(0)\rangle + |1, f(1)\rangle).$$

It seems that just by calling the gate once, the value of $f(0)$ and $f(1)$ can be computed. Though, this does not mean that we can get $f(0)$ and $f(1)$ simultaneously. As mentioned before, the only way to extract information from a quantum state is through a *measurement.* These measurements will not allow us to compute $f(0)$ and $f(1)$ together.

Exercise 7. Read about superposition on internet.

It might seem that the principle of superposition (allowing linear combinations of states) is not of much use, we can apply the function on multiple inputs but can only get the answer on one input.

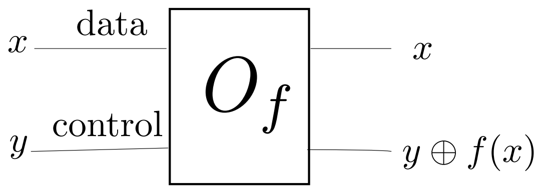

Fig. 1. Quantum implementation of computing a function, oracle for f

Exercise 8. Show that by applying Hadamard to second qubit, oracle for f gets converted from $|x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$ to $|x, y\rangle \rightarrow (-1)^{y \cdot f(x)}|x, y\rangle$.

Note 2. We have two possible oracles now to get the value of f, the first one computes value as XOR, the one in the exercise above computes the value in phase.

The trick to use superposition lies in *using* these hidden values of $f(0)$ and $f(1)$ for constructive/destructive interference.

Exercise 9. What will happen if we feed the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ as the data qubit and $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ as the control qubit to the oracle given above?

As promised, let us finally see the example where superposition can help us perform better than a classical computer.

1.1 The algorithm

We will describe *Deutsch's algorithm* in this section, which will show that a quantum computer can do interesting tasks not possible on a classical computer.

Suppose there is a function f from $\{0, 1\}$ to $\{0, 1\}$. We are given a gate (oracle, either XOR or phase) which can give us $f(b)$, given an input b. Deutsch's problem is to find whether $f(0) = f(1)$ and use minimum number of applications to this oracle computing f. For a classical computer, clearly we need two applications of the gate. In other words, it has to compute both $f(0)$ as well as $f(1)$ to answer Deutsch's question.

Exercise 10. Should we give the classical implementation of the function f to the quantum computer or the quantum implementation?

We should give the quantum implementation, the reason will be clear later in this course.

Let us take the quantum implementation of the function discussed in the previous section.

Exercise 11. Before looking at the algorithm, think about the following questions.

1. What should be the input for data qubit?
2. Which oracle should we use, the one with *XOR* or one where f is computed in phase (as power of -1)?
3. How can the output help?

There is an intuitive way to look at the algorithm. We can start in state $|+\rangle$ and compute the function over it. Since the values of $f(0)$ and $f(1)$ are needed to create interference, it might be better to compute the value of the function in phase. In other words, if $f(0) \neq f(1)$, there will be a relative phase between $|0\rangle$ and $|1\rangle$ at the date qubit. That means, we should have $|+\rangle$ state when $f(0) = f(1)$ and $|-\rangle$ otherwise.

Exercise 12. How can we distinguish between these two states?

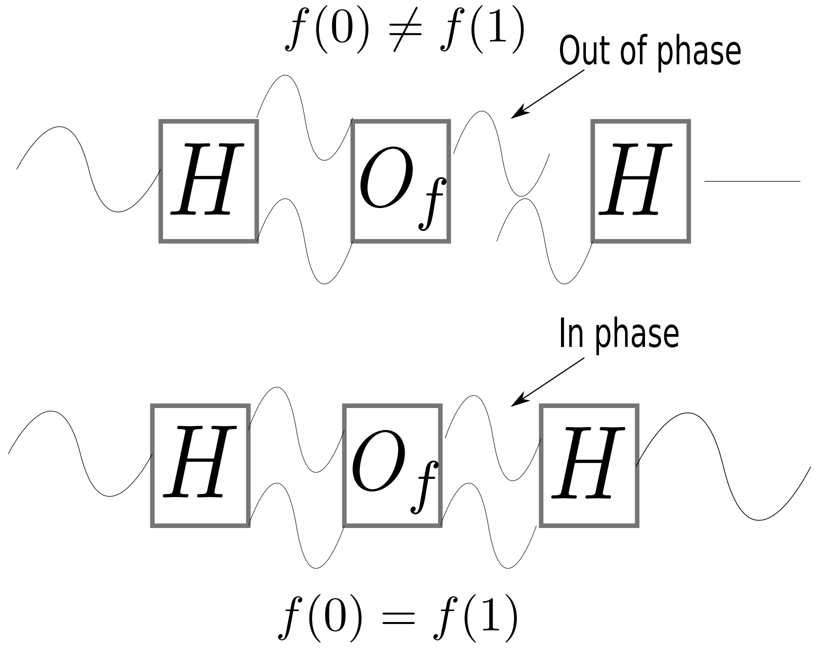

Fig. 2. Intuition for Deutsch's algorithm

The Figure 2 gives the idea, now look at Figure 3. Can you see why it is the implementation of the intuition?

Exercise 13. What is the purpose of the Hadamard gate to the control qubit?

Remember that the Hadamard gate on the control qubit (below one) is only meant to convert the XOR oracle into phase oracle. We can assume that the oracle is a phase oracle and remove the Hadamard gate to the control qubit. Then, our circuit is just Hadamard on the data (first) qubit, then oracle, then Hadamard again on the data qubit.

First Hadamard creates an equal superposition of two *waves*, one in state $|0\rangle$ and other in state $|1\rangle$. If $f(0) = f(1)$, then these two waves stay *in phase* even after the application of the oracle. If $f(0) \neq f(1)$, then these two waves go *out of phase* after we apply the oracle. The action of next Hadamard (on data qubit) is similar to having an interference of these waves. If they are in phase, then there is *constructive interference* and the amplitude of $|0\rangle$ is 1. If the waves are out of phase, there is *destructive interference* and the amplitude is 0 at $|0\rangle$. Given that Deutsch was a physicist, he might have thought of his algorithm in this way.

Let us look at the computation done by circuit in Figure 3 formally.

We will set the data qubit to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and control qubit to $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. To ease the notation, we will use subscripts, data qubit by $|\rangle_D$ and control by $|\rangle_C$. The initial state can be written as,

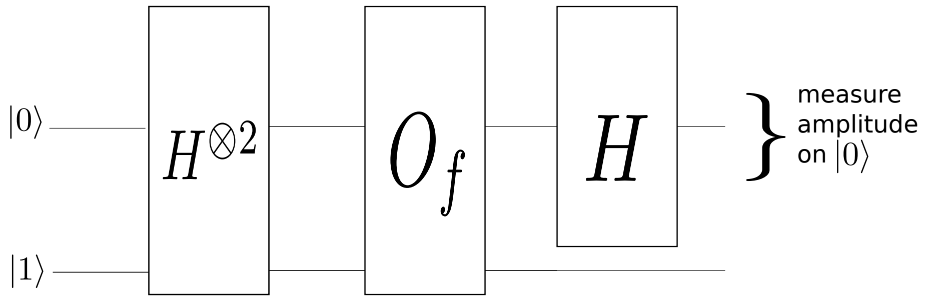

Fig. 3. Circuit for Deutsch's algorithm

$$\frac{1}{\sqrt{2}}(|0\rangle_D + |1\rangle_D)\frac{1}{\sqrt{2}}(|0\rangle_C - |1\rangle_C) = \frac{1}{2}(|0\rangle_D(|0\rangle_C - |1\rangle_C) + |1\rangle_D(|0\rangle_C - |1\rangle_C)).$$

After applying the function,

$$\frac{1}{2}(|0\rangle_D(|f(0)\rangle_C - |1 \oplus f(0)\rangle_C) + |1\rangle_D(|f(1)\rangle_C - |1 \oplus f(1)\rangle_C)).$$

If $f(0) = f(1)$ then the first qubit (data) will be in the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, otherwise it will be in state $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Observe that these states are the output of the Hadamard gate.

Remember that a quantum gate is always reversible. So, there exists an inverse of Hadamard gate. So we will apply the inverse of Hadamard gate, which is actually Hadamard gate itself, to the first qubit. We get $|0\rangle$ if $f(0) = f(1)$ and $|1\rangle$ otherwise.

By measuring the first register, we have solved the problem using just one query to the function. This shows that a quantum computer is faster than a classical computer for the Deutsch's problem.

The output of the function f has four possible values, $(00, 01, 10, 11)$. Classically, it was easy to distinguish whether f is $(00, 01)$ or $(01, 10)$. Using quantum mechanics, we were able to distinguish between $(00, 11)$ and $(01, 10)$. The amount of information gathered using one query to the black-box is only 1 bit. Surprisingly, that information (property of f) is not accessible by a classical computer.

You might complain that the problem seems very simple and the speed-up of 2 is not that impressive. This was just a toy example to illustrate how quantum property of superposition can be helpful. Throughout the course, there will be much more interesting problems and more impressive solutions.

2 Classical circuits

We have talked about the postulates of quantum mechanics; the other thing we need to understand is computation. We will assume that the reader is familiar with the very basic concepts in computation, like asymptotic notation and importance of Turing machine. If not, please look at the third chapter of Nielsen and Chuang [2] or the book by Arora and Barak [1].

Classically, computation is generally described by what can or cannot be performed on a Turing machine. Deutsch gave the concept of quantum Turing machine in 1985. Though, researchers in quantum computing prefer another equivalent model (to Turing machines) known as circuit model. We will introduce circuit model of quantum computation in this chapter, i.e., computation through quantum circuits. Let us look at classical circuits first.

We again assume that the reader is familiar with boolean circuits used to compute functions (circuits constructed from *AND*, *OR* and *NOT* gates). A circuit computes some function $f : \{0,1\}^n \rightarrow \{0,1\}^m$. In general, a circuit is a collection of gates connected by wires to transfer the information. A gate is a function from $\{0,1\}^k$ to $\{0,1\}^l$.

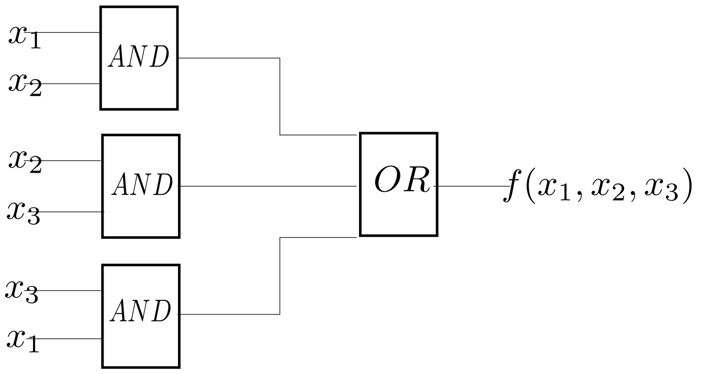

Fig. 4. A circuit computing a simple function. This one processes input from left to right, you might have seen one where it might go from bottom to top.

Exercise 14. What function does the above circuit compute?

Note 3. If there is a cycle in the circuit then the corresponding wire can have conflicting values. So, we assume that there are no cycles in the circuit. In other words, we can think of it as a directed acyclic graph. We will draw the circuits from left to right, there won't be any wires running from right to left to avoid cycles in the circuit.

Exercise 15. Draw a circuit for N*AND* on three bits using *AND*, *OR* and *NOT*.

In case of classical circuits, we allow an operation called *FANOUT*. This FANOUT operation has one input and two output; it makes two copies of the given input. You can easily show that copying is not a unitary operation, and hence not possible for quantum circuits.

What functions should we allow as part of a circuit? Clearly, if we allow arbitrary functions in the circuit then we can calculate arbitrary functions. Practically, it makes sense to assume some standard gate set and ask what functions can be computed from that gate set. As you know, one of the famous gate set is *AND*, *OR* and *NOT*.

Using the truth table, you can show that any function can be computed using the gate set of *AND*, *OR* and *NOT* (assignment). *AND*, *OR* on multiple bits can be broken down into *AND* and *OR* on two bits. Hence, we can have circuit for any function using *NOT* gate (on one bit) and *AND*, *OR* (on two bits). Even the *OR* gate can be computed with other two gates (De Morgan's rule), so *AND* and *NOT* are enough to perform any computation. Such a set of gates, which can compute any function, is called a *universal* gate set.

Exercise 16. Show that the gate set $\{NAND\}$ is universal.

We also allow creation of *ancilla* bits, these are extra bits required to do the calculation (workspace) and generally assumed to start in the state 0 ($|0\rangle$ for quantum circuits) of the required dimension. You can think of them as temporary variables in usual computation. Ancilla bits are important in quantum circuits specially because we need the computation to be reversible. Though, they need to be handled carefully, because they could get entangled with the output. More about that when we learn about quantum circuits.

2.1 Uniformity

We mentioned that the Turing machine model is equivalent to the circuit model. What does that mean? At a first look, Turing machines compute languages and circuits compute functions.

Exercise 17. What is the difference?

In case of languages, the input size is not fixed. For a language, the set of accepted strings is some subset of $\{0,1\}^*$. Alternatively, in case of circuits, the input size is fixed and hence we cannot talk about arbitrary size of strings.

One possible resolution is to have circuits for every length of input. That means, for every n there is a circuit C_n which accepts all strings of length n (or say less than n) and nothing else. Then, we can say that the circuit family $\{C_n\}$ computes the language.

Exercise 18. What function is the circuit computing in terms of the language?

This definition turns out to be very strong. Such family of circuits can potentially calculate by having the output hard coded for particular n's.

To take an example, let S be the set of strings s for which s_i is 1 iff i-th Turing machine (in some list) halts. Can we compute the language represented by set S? This is an undecidable problem for a Turing machine. Though, there is just one string of a particular length and a very small circuit for a particular length exists. In other words, the above definition of circuits will be able to compute an undecidable language!

The previous example shows that we should not be allowed to have very different circuits for different size of strings. This restriction, to avoid having arbitrary circuits for a particular size, is known as *uniformity.*

A circuit family is called uniform, if there exists a Turing machine which can generate circuit C_n, given an input n using bounded resources. Here, bound on resources is determined by the complexity class of interest. This circuit C_n should accept only and all the strings in the language of length less than n. Then, we say that this uniform circuit family $\{C_n\}$ computes the language.

Turing machine model and uniform circuit family is equivalent in the sense that whatever language can be computed in one model can be computed in another model too. This equivalence holds for quantum Turing machine and circuits too.

We haven't looked at the matter of efficiency. In general, an algorithm is considered *efficient* on a Turing machine if the output is determined in polynomial time. For the circuit family $\{C_n\}$ to be efficient,

– the Turing machine outputting circuits C_n (uniformity) should do so in polynomial time,
– the circuits should be of polynomial size in n (number of standard gates).

The first condition is mostly implicit in our algorithms; this is because circuits for different sizes are similar in our constructions. While designing and reading about algorithms, we focus on the second constraint for efficiency.

Let us see how quantum circuits are similar to classical circuits (and different from classical circuits).

3 Quantum circuits

We saw that classical circuits are made up of classical gates and wires which carried bits. Quantum circuits are analogous; they have quantum circuits with wires which carry quantum states or qubits. Through the postulates, we know that the quantum gates are unitary. So, quantum circuits will have unitary gates and measurements.

For example, look at the circuit given in Fig. 5. Notice that the wires run from left to right and there is no cycle. There are two inputs (x_1 and x_2) and two ancilla bits (y_1 and y_2). We have applied multiple gates, for example, Hadamard on the first qubit and CNOT's. A multi qubit unitary (whose description might be given separately, think of a subroutine) is applied on all 4 qubits. Finally, the output is obtained on the third qubit. The *meter* symbol represents measurement on the third qubit.

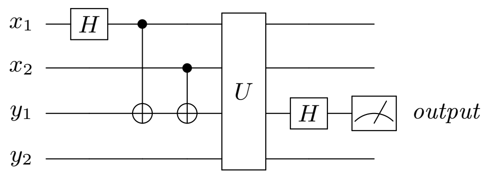

Fig. 5. An example of a quantum circuit

In general, it is sufficient to give the action of a circuit/gate on a basis. Choosing standard basis, it is enough to specify the action on classical bits. The action on other states can be inferred by linearity.

Exercise 19. What will be the action of $CNOT$ gate on state $\alpha_1|00\rangle + \alpha_2|01\rangle + \alpha_3|10\rangle + \alpha_4|11\rangle$ (first qubit is the control qubit)?

You should convince yourself that the answer is $\alpha_1|00\rangle + \alpha_2|01\rangle + \alpha_3|11\rangle + \alpha_4|10\rangle$.

3.1 Example of a quantum circuit

Let us try to construct a quantum circuit for the addition of two binary strings.

Exercise 20. Make sure that you can add two numbers given in binary representation (WITHOUT converting them into decimal representation). What is the sum of 10110 and 10011? You should get 101001.

Like the case of decimal representation, the trick is to add the numbers coordinate-wise and move the carry forward. For example, let a=1011 and b=1101, then

$$\begin{array}{r|l} \textit{carry} & 1\,1\,1\,1 \\ \hline a = & \;\;1\,0\,1\,1 \\ b = & \;\;1\,1\,0\,1 \\ \hline \textit{ans} = & 1\,1\,0\,0\,0\,0 \end{array}$$

Looking at this procedure, one of the main part (subroutine) seems to be the addition of two bits. There will be two output, one bit goes to the answer and other is the carry. Such a circuit is called a *half adder.* To make it precise, here is the output on all 4 possible classical inputs (the most significant bit is the carry).

$$\begin{array}{c} 00 \to 00 \\ 01 \to 01 \\ 10 \to 01 \\ 11 \to 10 \end{array}$$

It is not very difficult to extend the half adder circuit to addition of two full binary strings.

Exercise 21. Can you construct a circuit to do addition of two 2-bit strings, given half adder as a blackbox? Can you generalize it to n-bit strings?

Try to finish the circuit by yourself. If not, look at Fig. 7. For the quantum case, answer needs to be given on two ancilla qubits. A quantum half adder will have two inputs and two ancilla bits, for the sake of clarity, Fig. 7 only shows two inputs bits on the left hand side and two ancilla bits on the right hand side.

Our remaining task in this section is to figure out a circuit for half adder. We will make it *bit by bit.* For the first bit (one that goes to the answer), it is 1 if and only if exactly one of the two input bits is 1.

Exercise 22. Do you know the name of this gate in the classical world?

It is called an *XOR* operation. What is the quantum equivalent? You have seen that gate before, it is the *CNOT* gate (with *XOR* computed in date qubit). This would have worked if we wanted the answer qubit as one of the input qubits. What if we want to compute answer (and carry) on an ancilla qubit starting in $|0\rangle$ state?

Exercise 23. How can you compute the answer in an ancilla qubit?

Since the ancilla is set to 0, you can take *XOR* with the first qubit and then the second qubit. In other words, use two *CNOT* gates.

We are left with the task of computing the carry bit. It is 1 if and only if both the qubits are 1. This is known as *AND* gate classically.

Exercise 24. Use Toffoli (CCNOT) gate to create a circuit for *AND* gate.

The final circuit will look like the following figure.

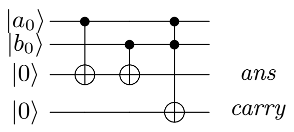

Fig. 6. Circuit for half adder

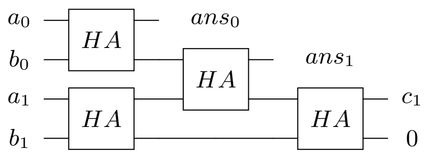

Fig. 7. Circuit for addition using half adder (HA). The top bit of HA is the answer bit and the bottom one is the carry.

3.2 Gates of a quantum circuit

The main difference is the kinds of gates applied in a quantum circuit, remember that they have to be unitary. The simplest unitary gates are single qubit gates. We have already looked at Pauli matrices (X, Y, Z) and Hadamard gate.

Exercise 25. Do you remember the definition of these gates?

It is sometimes easier to remember gates by their action instead of their matrix representation. The action needs to be specified only on the basis states.

Pauli X is like a NOT gate, it interchanges states $|0\rangle$ and $|1\rangle$. Pauli Z puts a phase of -1 if the state is $|1\rangle$ and does nothing on state $|0\rangle$. Pauli Y can be obtained by the relation $Y = iXZ$. The Hadamard gate moves $|0\rangle$ to $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|1\rangle$ to $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ states.

Another important gate is the phase gate, which puts a phase of i if the qubit is $|1\rangle$ (it is the square root of Z gate). The unitary matrix S for the phase gate is given by,

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

Another important gate is $\frac{\pi}{8}$ gate. It is the square root of phase gate.

Exercise 26. Show that the following gate is a square root of the phase gate.

$$\begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix}$$

Exercise 27. Suppose A is the Hermitian matrix and x is a real number, s.t., $A^2 = I$. Show that $e^{iAx} = cos(x)I + isin(x)A$.

Exercise 28. Calculate $R_X(\theta) = e^{-iX\theta/2}, R_Y(\theta) = e^{-iY\theta/2}, R_Z(\theta) = e^{-iZ\theta/2}$. It can be shown [2] that any single qubit unitary operation can be decomposed in terms of these rotations and a global phase shift.

What about gates on more than one qubit? We need an important class of operators called controlled operators. We have already seen $CNOT$ gate. In terms of computational states, the action of $CNOT$ can be written as,

$$|x\rangle|y\rangle \to |x\rangle|y \oplus x\rangle.$$

It turns out that controlled version of any single qubit gate (e.g. CZ gate) can be implemented using $CNOT$ and other single qubit gates. Actually, any unitary can be performed given $CNOT$ and all single qubit gates [2].

We can also define controlled operations where we have multiple control qubits. The Toffoli gate is an important example with two control qubits. It is also called $CCNOT$ gate. It applies an X gate (NOT) iff both control qubits are set to $|1\rangle$.

Exercise 29. Write the matrix representation for Toffoli gate.

Another important part of the quantum circuits is measurement. We state two properties of measurement without proof, interested readers can refer to Nielsen and Chuang [2].

1. In general, there can be measurements in between and the classical information can be used to control further elements in the later part of the circuit. Even if we replace the classically controlled operations by quantumly controlled operators, the final output is the same. To summarize, we can assume that the measurements are done at the end of the circuit.
2. We can assume that all un-terminated wires at the end are measured. It does not affect the measurement statistics of the remaining system.

Like the case of classical circuits, we know of combinations of gates which are universal for quantum circuits (though the meaning of universality is slightly different). For our purpose it is enough to assume that we can use all the standard gates mentioned till now in our circuits without worrying about the universality. Also, it is possible to simulate any classical circuit using a quantum circuit (without much change in size). So, we can assume access to not only standard quantum gates but also any classical circuit (with equivalent cost). There is some more information in the two subsections (Section 3.3 and Section 3.4) below, interested reader can go through them (and Nielsen and Chuang's book [2]) for more details.

3.3 Universality

We saw that *AND*, *OR* and *NOT* gates were universal for classical circuits. Can we come up with universal gate sets for quantum circuits? It was mentioned that $CNOT$ and all single qubit unitary can perform any unitary operation. Though, this does not seem to be a practical choice (because of the size of this gate set).

A universal set of gates G in quantum computation are those which can approximate any unitary operator with arbitrary accuracy. In other words, for any unitary U and for arbitrary accuracy, there is a circuit with gates from G implementing U with that accuracy.

It turns out that the set of Hadamard, $CNOT$ and $\frac{\pi}{8}$ gates is universal in this sense [2]. Again, it does not tell us about the size of the circuit needed to compute U. We can only use those U's, which can be implemented using only *polynomial* number of implementations of *basic* gates (gates from universal set of families).

This brings up another question. There are multiple families of universal gate sets for quantum computation. What if some operation can be performed using polynomial applications of one gate set but not with another?

Exercise 30. Why is this not a problem in classical case?

Solovay-Kitaev theorem tells us: Given a universal gate set G, any 1 or 2 qubit unitary can be performed with accuracy ϵ using only $polylog(\frac{1}{\epsilon})$ number of gates from G. Intuitively, any 1 or 2 qubit unitary can be simulated with constant (assuming accuracy to be constant) number of gates. So, it doesn't matter which set of universal gates we pick.

To summarize, we have a large class of gates at our disposal.

– All standard gates discussed till now. This includes Pauli gates (X,Y,Z), Hadamard, CNOT, CCNOT.
– All classical circuits with the same complexity up to a constant.
– Any 1 qubit gate or controlled 2 qubit gate can be implemented approximately (Solovay-Kitaev), this can be generalized to any constant qubit gate.
– Any subroutine given to us with the required complexity.

3.4 Extra reading: Classical circuit simulation

We have seen classical circuits as well as quantum circuits. It was mentioned before that anything which can be done on a classical computer can be simulated on a quantum computer. At a glance, it seems difficult; especially since *AND*, *OR* and *NOT* are irreversible, but quantum gates are constrained to be reversible.

It turns out that we can have reversible versions of *AND*, *OR* and *NOT* gates constructed from Toffoli gate. Just a reminder, a Toffoli gate is a $CCNOT$ gate with the following action on basis elements,

$$a, b, c \rightarrow a, b, c \oplus ab.$$

Exercise 31. Show that the Toffoli gate is reversible.

You can easily show that the Toffoli gate can be used to simulate N*AND* and FANOUT gate. For example, inputting $a, b, 1$ will simulate the N*AND* gate. This is not enough, we need to simulate FANOUT gate too !

Exercise 32. Show that you can simulate FANOUT gate using Toffoli gate. Why is this not against *no cloning* theorem?

Hence, any arbitrary classical circuit can be simulated by an equivalent reversible circuit. A circuit from Toffoli gates can be thought of as a quantum circuit, by extending its action on all states using linearity. To conclude, any classical circuit can be simulated by a quantum circuit of equivalent circuit size (up to polynomial factors).

For the randomized computation, we also need to produce random bits. This can be done by applying Hadamard on $|0\rangle$ and then measuring the qubit in the standard basis. This shows that any randomized computation can also be efficiently simulated by a quantum circuit.

Suppose, a classical black-box outputs $f(x)$ on an input x. This means, at the time of implementation of the black-box, there will be some classical circuit which will compute $f(x)$ using x. This circuit can be converted into a reversible circuit which acts as $(x, y) \to (x, y \oplus f(x))$. Hence, the quantum version of the blackbox will be the circuit that acts similarly on the basis states, and acts linearly on the other states.

This conversion is of great importance. Suppose we apply Hadamard on the input qubit x (required tensor product if the input is multiple qubits) with the second qubit $|0\rangle$. What is the output of the blackbox? You will answer this as part of the assignment.

Suppose x was only 1 bit then the output is,

$$\frac{1}{\sqrt{2}}|0, f(0)\rangle + |1, f(1)\rangle.$$

In some sense, we have computed the function on all possible inputs.

This aspect is known as *quantum parallelism*. The catch is, the only way to get information from this state is by a measurement. This implies, we can only get the output for one particular input.

Note 4. We can only compute $(x, y) \to (x, y \oplus f(x))$, that does not imply $x \to f(x)$.

4 Assignment

Exercise 33. A half-adder is a circuit which takes x, y as input and gives output $x \wedge y, x \oplus y$. Draw a circuit for half adder using *AND*, *OR* and *NOT*.

Exercise 34. Make a circuit which takes x, y, c as input (interpret c as carry) and output $c', x \oplus y \oplus c$ where c' is one if at least two of x, y, c are 1. Why is this circuit helpful.

Hint: You can use half adder from the previous exercise.

Exercise 35. Create a circuit to convert $|00\rangle$ to equal superposition of all 4 standard basis states. Can you extend this argument to creating equal superposition over all 2^n standard states using appropriate input.

Exercise 36. Show that *AND*, *OR* and *NOT* form a universal gate set.

Exercise 37. Consider the oracle/blackbox for a function as defined in the text. Suppose we apply Hadamard on the input qubit x (required tensor product if the input is multiple qubits) with the second qubit $|0\rangle$. What is the output of the blackbox?

Exercise 38. Show that SWAP (interchanging two qubits) can be performed using CNOT. Hint: how would you do it classically without any extra variable (memory)?

Exercise 39. What will be the action of the following circuit on the input $|++\rangle$ state.

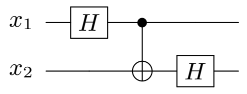

Fig. 8. A quantum circuit

References

1. S. Arora and B. Barak. *Computational Complexity: a modern approach.* Cambridge, 2009.
2. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information.* Cambridge, 2010.