CMPT 409 | 2022 Summer | Assignment 3

Analysis of Deutch’s Algorithm in Quantum Circuit

July 14th, 2022

Group Quanteam
Jundi Chen #301387982
Navjot Kaur #301404765
Revika Jain #301369220
Rahil Sharma #301401661
Yeoran Kim #301422131

1. Introduction

Deutch's algorithm was introduced in 1985 and is the first example that proved the power of quantum computing. A British physicist, David Deutch, showed that some problems could be solved exponentially faster than any classical algorithm by utilizing this quantum algorithm.

To introduce the algorithm briefly, consider a black box containing the function f(x). f(x) takes a string of single bit, {0, 1} as input, and the output range is the same as {0, 1}. There are four possible arrangements for this function.

$f_i(x)$	$x = 0$	$x = 1$	$f_i(0) \oplus f_i(1)$
f_0	0	0	0
f_1	0	1	1
f_2	1	0	1
f_3	1	1	0

If f(x) produces the same output for any input, f(x) is said to be a constant function; otherwise, f(x) is a balanced function. Deutsch-Jozsa algorithm that extends the Deutsch algorithm with *n*-bit of strings instead of the single bit was also proposed later in 1992. This paper limits the discussion and implementation of Deutsch's algorithm only.

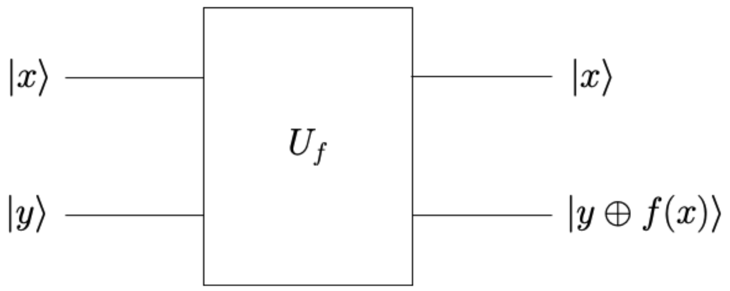
In classical computing, two evaluating queries are required for each input to determine if f(x) is either constant or balanced correctly. All inputs in {0, 1} should be checked in this case. The cost of queries grow exponentially once the input range gets larger.

Deutsch's algorithm in quantum computing needs only one evaluation for the correct answer in a quantum circuit, regardless of the input size. Moreover, the generalized version of Deutsch's algorithm demonstrates that quantum computing could achieve an exponential order of efficiency compared to classical computing. It is because of the principle of quantum superposition, which states that a general qubit state is a combination of all possibilities by utilizing the property of quantum interference.

The following sections discuss how the algorithm achieves the speed-up in a quantum circuit in detail, actual code implementations in Qiskit, and compare the algorithm with and without a phase kickback.

2. The Problem Statement In Quantum Computing

The problem that Deutsch's algorithm tackles is stated in terms of binary functions f: $\{0, 1\} \rightarrow \{0, 1\}$. Based on Postulate 2 of quantum mechanics, all quantum operations must be unitary and hence reversible. In general, however, given the output f(x) of a function, it is not always possible to invert f to obtain the input x. So, we have to compute f(x) in such a way to guarantee that the computation can be undone. This is achieved via the following setup:



Based on the above diagram, we can see that, U_f is a unitary operator mapping $|x > |y > \rightarrow |x > |x \oplus y >$ for any x, y $\in \{0, 1\}$, and where $\oplus$ denotes XOR or addition modulo 2 (Controlled NOT gate).

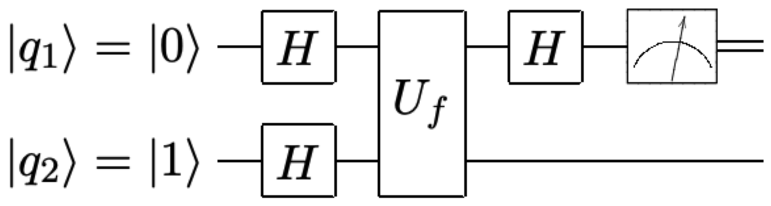
The problem Deutsch's algorithm tackles can now be stated as follows. Given a block box U_f implementing some unknown function $f\colon \{0, 1\} \rightarrow \{0, 1\}$, determine whether f is "constant" or "balanced". Here, constant means f always outputs the same bit, i.e. $f(0) = f(1)$, and balanced means f outputs different bits on different inputs, i.e. $f(0) \neq f(1)$.

Quantum Query complexity

The "cost function" we are interested to minimize in solving Deutsch's problem is number of quantum queries of U_f. This is an example of the model of quantum query complexity, in which many quantum algorithms have been developed. In the study of quantum query complexity, one is given a black box U_f implementing some function f , and asked what the minimum number of required queries is in order to determine some desired property of f . The quantum algorithm computing this property can consist of a lot of quantum gates , e.g 999999999 quantum gates. However, if it contains only 2 queries to U_f , then we consider the cost of the algorithm as 2, i.e. all "non-query" operations are considered free.

3. Deutsch's Algorithm in Circuit

The circuit for Deutsch's algorithm is give as follows:



It is not a priori obvious at all why this circuit should work, and this is indicative of designing quantum algorithms in general – the methods used are often incomparable to known classical algorithm design techniques, and thus developing an intuition for the quantum setting can be very difficult. Let us hence simply crunch the numbers and see why this circuit indeed computes $f(0) \oplus f(1)$, as claimed.

The circuit is divided into 4 phases, first is at the start of the circuit (|ψ1 >), next is after the first Hadamards are applied (|ψ2 >), next is after U_f is applied (|ψ3 >), and last is after the last Hadamard gate is applied (|ψ4 >). It is clear that

$$\begin{aligned} |\psi_1\rangle &= |0\rangle|1\rangle, \\ |\psi_2\rangle &= |+\rangle|-\rangle = \frac{1}{2}(|0\rangle|0\rangle - |0\rangle|1\rangle + |1\rangle|0\rangle - |1\rangle|1\rangle). \end{aligned}$$

After the oracle U_f is applied, we have state:

$$|\psi_3\rangle = \frac{1}{2}(|0\rangle|f(0)\rangle - |0\rangle|1 \oplus f(0)\rangle + |1\rangle|f(1)\rangle - |1\rangle|1 \oplus f(1)\rangle).$$

Before we apply the final Hadamard, it will be easier to break our analysis down into two cases: When f is constant and when f is balanced.

Case 1: Constant f. By definition, if f is constant, then $f(0) = f(1)$. Therefore, we can simplify $|\psi_3 >$ to

$$\begin{aligned} |\psi_3\rangle &= \frac{1}{2}(|0\rangle|f(0)\rangle - |0\rangle|1 \oplus f(0)\rangle + |1\rangle|f(0)\rangle - |1\rangle|1 \oplus f(0)\rangle) \\ &= \frac{1}{2}((|0\rangle + |1\rangle) \otimes |f(0)\rangle - (|0\rangle + |1\rangle) \otimes |1 \oplus f(0)\rangle) \\ &= \frac{1}{2}(|0\rangle + |1\rangle) \otimes (|f(0)\rangle - |1 \oplus f(0)\rangle) \\ &= \frac{1}{\sqrt{2}}|+\rangle \otimes (|f(0)\rangle - |1 \oplus f(0)\rangle). \end{aligned}$$

Thus, qubit 1 is now in state $| +>$. We can conclude that:

$$|\psi_4\rangle = \frac{1}{\sqrt{2}}|0\rangle \otimes (|f(0)\rangle - |1 \oplus f(0)\rangle),$$

i.e. qubit 1 is exactly in state $|0>$. Thus, measuring qubit 1 in the standard basis now yields outcome 0 with certainty.

Conclusion If f is constant, the algorithm outputs 0, if f is balanced, the algorithm outputs 1. Thus the algorithm decides whether f is constant or balanced, using just a single query.

4. The Phase Kickback Trick

To analyze Deutsch's Algorithm using phase kickback, just before U_f is applied, we will return to the following state.

$$|\psi_2\rangle = |+\rangle|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle|-\rangle + |1\rangle|-\rangle).$$

After the application of phase kickback equation, we know that we will have the following state after U_f is applied

$$|\psi_3\rangle = \frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle|-\rangle + (-1)^{f(1)}|1\rangle|-\rangle).$$

So suppose that f is constant in other way we can write, $f(0) = f(1)$, and now to simplify $|\psi_3>$ we will take out the -1 phase factor to have,

$$|\psi_3\rangle = (-1)^{f(0)}\frac{1}{\sqrt{2}}(|0\rangle|-\rangle + |1\rangle|-\rangle) = (-1)^{f(0)}|+\rangle|-\rangle.$$

Then, by applying the final Hadamard to qubit 1 gives

$$|\psi_4\rangle = (-1)^{f(0)}|0\rangle|-\rangle.$$

The first qubit may now be measured with certainty to provide the same result, 0, as before. If $f(0) \neq f(1)$, or f is balanced then we cannot simplify the way we did previously by factoring out -1.

Hence, up to a factor of ±1 overall we get,

$$|\psi_3\rangle = \pm\frac{1}{\sqrt{2}}(|0\rangle|-\rangle - |1\rangle|-\rangle) = \pm|-\rangle|-\rangle.$$

5. The Algorithm Implementation in Qiskit

This code is implemented using Qiskit on jupyter lab in Python. We first imported the required libraries such as QuantumCircuit, QuantumRegister, ClassicalRegister, etc from qiskit. The matplotlib library is used for the visualisation of the circuits. The plot_histogram library is used to create histogram of the probabilities of the result that indicates whether the f function is balanced or not. The code snippet is provided below.

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit import BasicAer
from qiskit import execute
from qiskit.visualization import plot_histogram
import matplotlib
```

The code snippet provided below denotes the Oracle function that calculates the balance and constant functions. Here as we know for the balance part there are two possibilities 01, and 10. We use CNOT gate (qc1.cx(0,1) in code) for qubit in register 1, with the qubit in register 2 as target. For the constant part we have two possibilities 00, and 11. Here only the input is used to calculate the constantness.

```
def balance1(qc1):
    qc1.i(1)
    qc1.cx(0, 1)
    qc1.barrier()
    return qc1

def balance2(qc1):
    qc1.x(1)
    qc1.cx(0, 1)
    qc1.barrier()
    return qc1

def const1(qc1): #f(x) = 1
    qc1.x(1)
    qc1.barrier()
    return qc1

def const2(qc1): #f(x) = 0
    qc1.i(1)
    qc1.barrier()
    return qc1
```

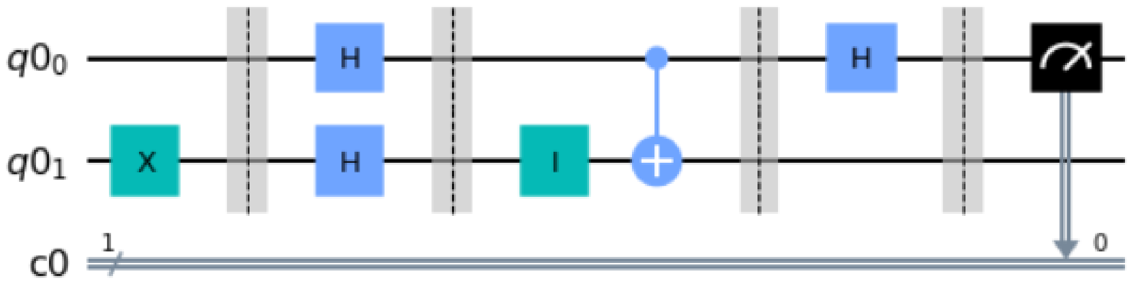
Let us move on to writing the Deusch function. We use 2 qubits and one classical bit to create our quantum circuit. We start by adding a single input value x with an initial state of 1. Then we apply a barrier in order to separate the circuit phase. Now we apply hadamard on both qubits. This brings our quantum state in a superposition of all possible 2 qubit states: 00,01,10,11. We now need to apply a reversible unitary operation (oracle) that will evaluate the result. Now we again apply a Hadamard gate to the input qubit x to make the circuit reversible. After the the result is measured, as can be seen from the code snippet provided below:

```
def deutsch(function):
    qr1 = QuantumRegister(2) #2 qubits
    cr1 = ClassicalRegister(1) #1 classical register
    qc = QuantumCircuit(qr1, cr1) #quantum circuit

    qc.x(1) #initial state
    qc.barrier() #barrier to separate the circuit phases

    qc.h(0) #hadamard gate for superposition
    qc.h(1) #hadamard gate
    qc.barrier()

    U = function(qc) #oracle (U) function

    U.h(0) #hadamard gate again
    U.barrier()

    U.measure(qr1[0], cr1) #measuring the result

    return U
```

Let us see a diagrammatic representation of the circuit by drawing it to check for balance function.

```
result1 = deutsch(balance1)
result1.draw(output='mpl')
```

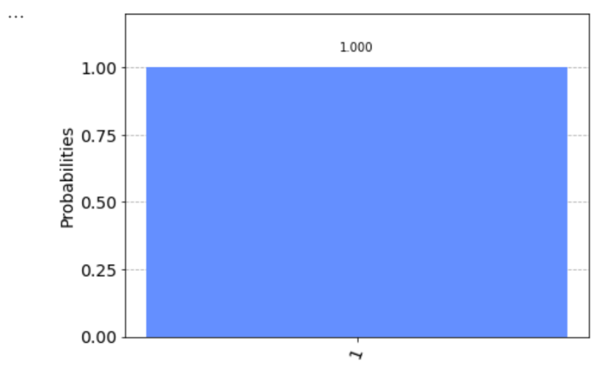
The output of the snippet is shown below:



To check if the result is balanced indeed we used the QasmSimulator to mimic an actual device, which executes the quantum circuit.

```
#QasmSimulator to mimic an actual device, which executes a Qiskit
QuantumCircuit
back = BasicAer.get_backend('qasm_simulator')
shots = 1024
answer1 = execute(result1, backend = back, shots=shots).result()
answer1 = answer1.get_counts()
plot_histogram(answer1)
```

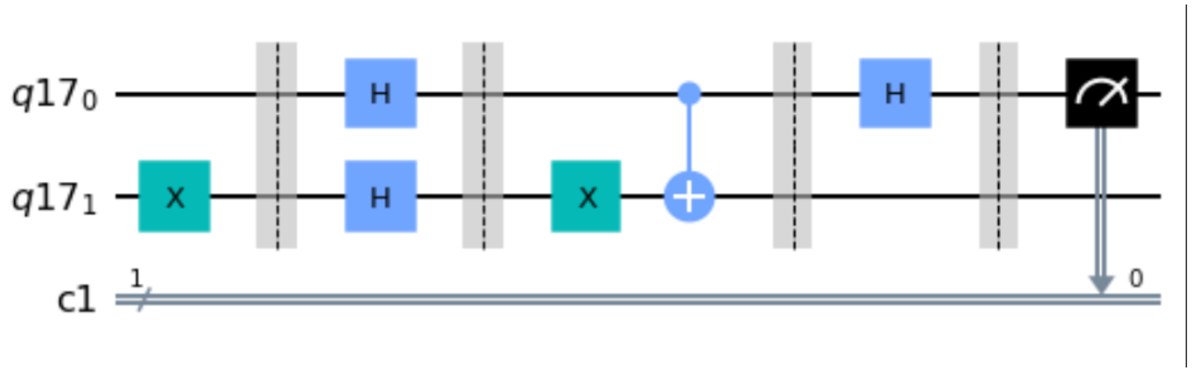
Here the result is 1 with the probability 1, so the function is balanced.



Let us check the other case of balance which is implemented as balance2 function.

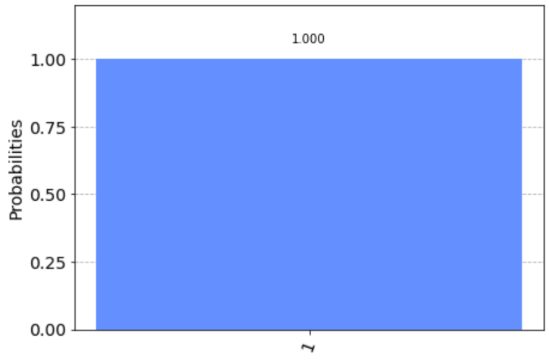
```
result2 = deutsch(balance2)
result2.draw(output='mpl')
```

Circuit:



```
answer2 = execute(result2, backend = back, shots=shots).result()
answer2 = answer2.get_counts()
plot_histogram(answer2)
```

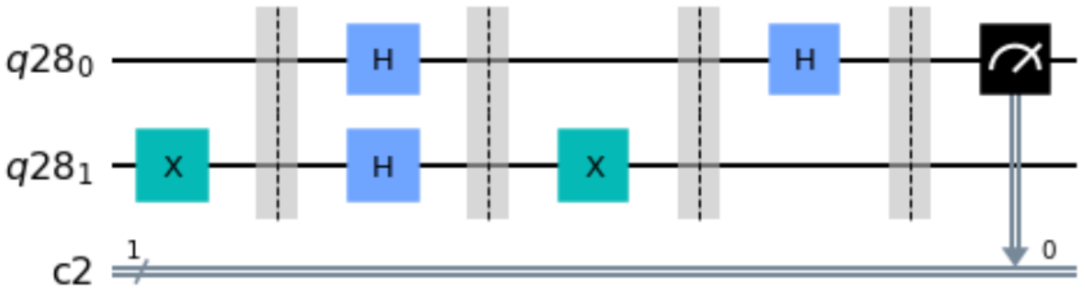
Here the result is 1 with the probability 1, so the function is balanced.



Now lets check for the constant function f(1, 1).

```
result3 = deutsch(const1)
result3.draw(output='mpl')
```

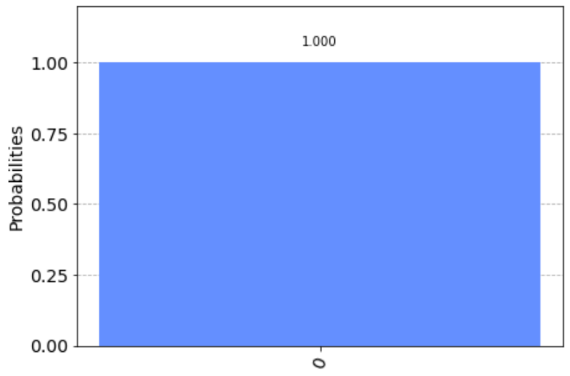
Circuit:



```
answer3 = execute(result3, backend = back, shots=shots).result()
answer3 = answer3.get_counts()
plot_histogram(answer3)
```

Here the result is 0 with the probability 1, so the function is constant.

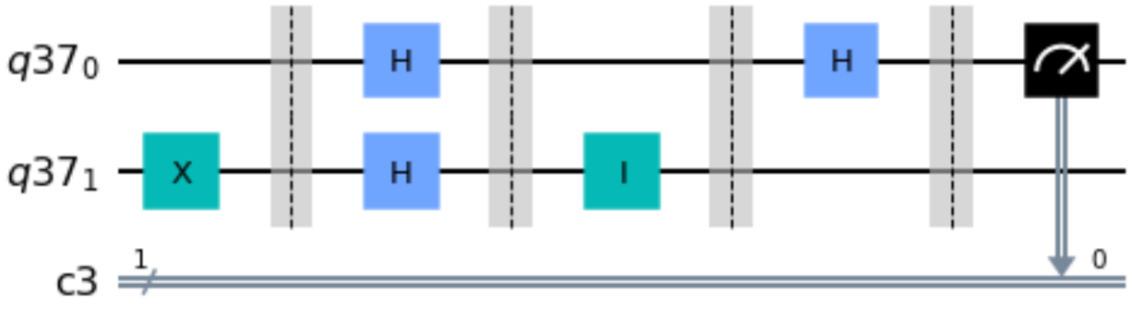
...



At last we will check for the other constant function f(0, 0)

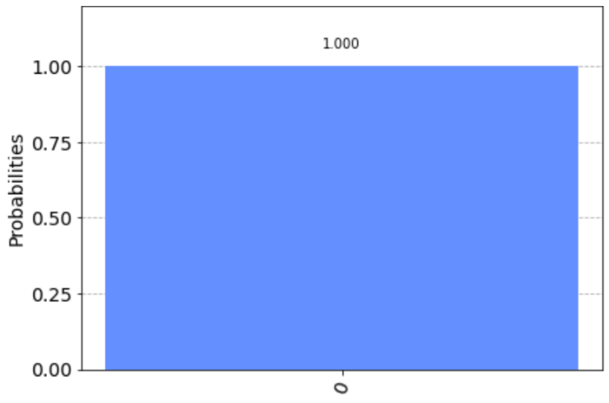
```
result4 = deutsch(const2)
result4.draw(output='mpl')
```

Circuit:



```
answer4 = execute(result4, backend = back, shots=shots).result()
answer4 = answer4.get_counts()
plot_histogram(answer4)
```

Here the result is 0 with the probability 1, so the function is constant.



6. References

[1] *Code example: Deutsch-Joza-algorithm*, Quantum Inspire. [Online]. Available: https://www.quantum-inspire.com/kbase/deutsch-jozsa-algorithm/
[2] *Deutsch-Joza-algorithm*, Wikipedia. July, 2002. [Online]. Available: https://en.wikipedia.org/wiki/Deutsch%E2%80%93Jozsa_algorithm
[3] *Deutsch-Joza-algorithm*, IBM Quantum Compose. [Online]. Available: https://quantum-computing.ibm.com/composer/docs/iqx/guide/deutsch-jozsa-algorithm
[4] *Deutsch-Joza-algorithm,* Qiskit. [Online]. Available: https://qiskit.org/textbook/ch-algorithms/deutsch-jozsa.html
[5] P. Yeong, *Deutsch-algorithm*. October 24, 2019. [Online]. Available: https://young.physics.ucsc.edu/150/deutsch.pdf
[6] S. Gharibian, *Lecture 6: Deutsch's Algorithm*, CMSC 491: Introduction to Quantum Computation Sevag - Fall 2015. [Online]. Available: http://www.people.vcu.edu/~sgharibian/courses/CMSC491/notes/Lecture%206%20-%20Deutsch's%20algorithm.pdf
[7] S. Pearce, *Introduction to Quantum Computing Weeks Seven and Eight*, CMPT 409 lecture - Summer 2022, p38-58.
[8] V. Onofre, *The Deutsch algorithm,* Full-Stack Quantum Computation. March 4, 2022. [Online]. Available: https://fullstackquantumcomputation.tech/blog/deutsch-algorithm/