Quantum Computing:

An Introduction

Introduction: Traditional Computing

In a traditional computer system all information is represented in bits. A bit can represent one of two values 1 or 0. Just like a light switch can only be one or off. Within a computer voltage is used to set bits on/off. With low and high voltages representing 0 and 1 respectively

Given a single bit A we can represent two states:

A
0
1

Given two bits A and B we can represent four states:

A	B
0	0
0	1
1	0
1	1

Introduction: Quantum Mechanics

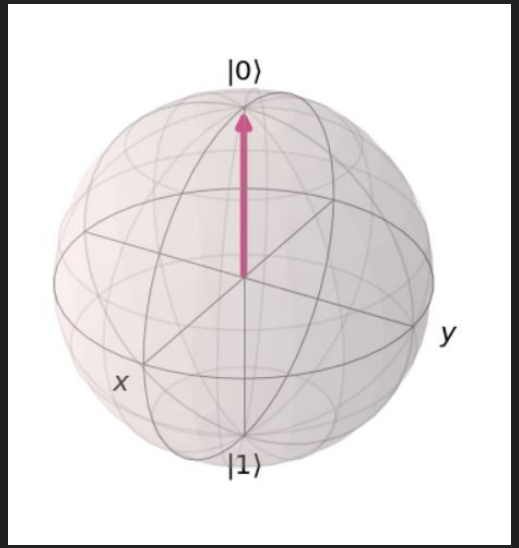
A Quantum Computer takes advantage of Quantum Mechanics in order to operate on Qubits or quantum bits.

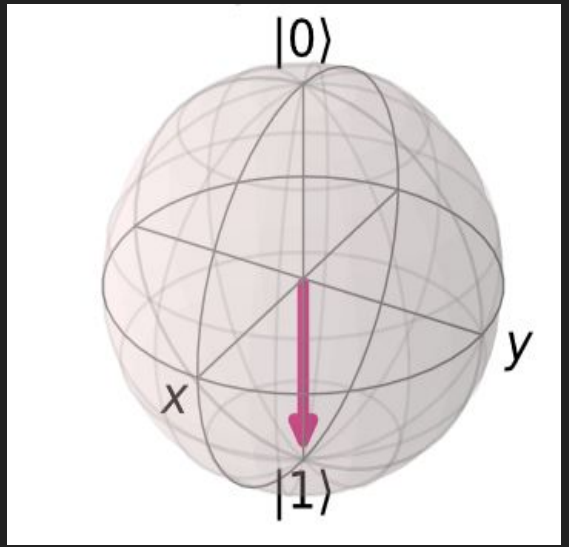
- Qubit: Takes advantage of superposition in order to represent more than just one state at a time.

- SuperPosition: Is best described through an example. Consider a coin being flipped in the air. While the coin has only two possible states, heads or tails, while the coin is flipping in air it's impossible to discern which state its in. Instead the coin is said to be in both states. That is to say the coin is in a SuperPosition of all its states.

Intro: Quantum Mechanics

More on Qubits: Where bits measure there state based on voltage. A quantum computer measures the state of a qubit based on its spin. With an up spin representing the pure state of 0, and a down spin represents the pure state of 1.





Qubits: Ket Notation

The best way to represent Qubits is through the languages of linear algebra and probability. As an example we can represent any single Qubit in “ket” notation:

- Given a Qubit a:

$$|a\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

State:

0

1

Where a is a 2 by 1 vector representing the probability of qubit a being in each of its states.
In the case of a it has a 100% chance of being in its 0th state and 0% chance of being in its 1st state.

Qubits: Pure States

A qubit can be in either a superposition of its states, or in one of two pure states.

These are represented in ket notation as follows: | 0⟩ and | 1⟩

State:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \begin{matrix} 0 \\ 1 \end{matrix}$$

State:

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \begin{matrix} 0 \\ 1 \end{matrix}$$

Where | 0⟩ represents spin up (or zero state) and | 1⟩ represents spin down (or one state).

Matrix Operations:

Because Qubits are represented as 2 by one vectors, which are special cases of matrices, any operation performed on a qubits is equivalent to an operation on matrices.

<u>Matrix Multiplication:</u>
Given two matrices A and B, we multiply the columns of A by the rows of B and placing the corresponding sum in the new matrix, in order to multiply them together.

$A = \begin{bmatrix} 2 & 0 \\ 1 & -1 \end{bmatrix}$ $B = \begin{bmatrix} -3 & 1 \\ 2 & 1 \end{bmatrix}$ $AB = \begin{bmatrix} 2 & 0 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} -3 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} -6 & 2 \\ -17 & 4 \end{bmatrix}$

2(-3) + 0(2) = -6

Matrix Operations:

Scalar Multiplication:

Given a matrix A and a scalar (constant) value c we can perform scalar multiplication by multiplying every value in A by c:

Let A = $\begin{bmatrix} 2 & 0 \\ 1 & -1 \end{bmatrix}$ Let c = $\frac{1}{\sqrt{2}}$ cA = $\frac{1}{\sqrt{2}}\begin{bmatrix} 2 & 0 \\ 1 & -1 \end{bmatrix}$ = $\begin{bmatrix} \frac{2}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}$

Quantum operations:

Hadamard Gate:

A Hadamard gate H is one of the most fundamental operation in quantum computing. It puts two qubits into a state of superposition. The Hadamard gate can be expressed in matrix form as follows:

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

Quantum operations:

Pauli X-Gate:

The Pauli X-Gate is analogous to the traditional NOT Gate in that it switches the state of a pure state. For a traditional bit this is done by changing voltage, but it a qubit this is done by flipping the qubit about its x-axis.

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

Ex: The X gate operating on $|0\rangle$:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

Quantum Operations:

Z-Gate: The z-gate is similar to a x-gate. Where the x-gate flips a qubit about it x-axis, the z-gate rotates a qubit between its x, and z-axis.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Ex:

$$Z\,|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

Multiple Qubits:

Two represent multiple qubits we use tensor notation.

Consider two qubits |***a*** ⟩ and |***b***⟩ where:

|***a*** ⟩ = $\begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$ ***b***⟩= $\begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$

|***ab***⟩ = $\begin{bmatrix} a_0 * \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \\ a_1 * \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \end{bmatrix}$ = $\begin{bmatrix} a_0 b_0 \\ a_0 b_1 \\ a_1 b_0 \\ a_1 b_1 \end{bmatrix}$

Multiple Qubit Operations:

Performing operations on multiple qubit systems, requires tensor notation
Consider a two qubit system made up of

$|q_0\rangle$ and $|q_1\rangle$, and say we wish to perform $H|q_0\rangle$, and $X|q_1\rangle =$ $H \otimes X|q_0q_1\rangle$

$$H \otimes X|q_0q_1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 * \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & 1 * \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ 1 * \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & 0 * \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \end{bmatrix}$$

$$H \otimes X|q_0q_1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \\ 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & H \\ H & 0 \end{bmatrix}$$

Grover’s Algorithm:

One of the biggest advantages of a quantum computer over a traditional one is its ability to search at a much faster rate. Grover’s algorithm is a perfect example of a quantum search algorithm.

Suppose you had list of N items, identified with a unique number sorted in a random order. How would we find a specific number linking us to the item we want?

- Traditional Computer:
  - A traditional computer can only check one number at at time. Meaning on average you would have to search through half of the entries before you would the number you were looking for. In the worst case you have to search through all N items before you found the right number.
- **Quantum Computer:**
  - On a quantum computer we can take advantage of superposition and probability to complete the search in a total of $\sqrt{N}$ steps.

Grover's Algorithm:

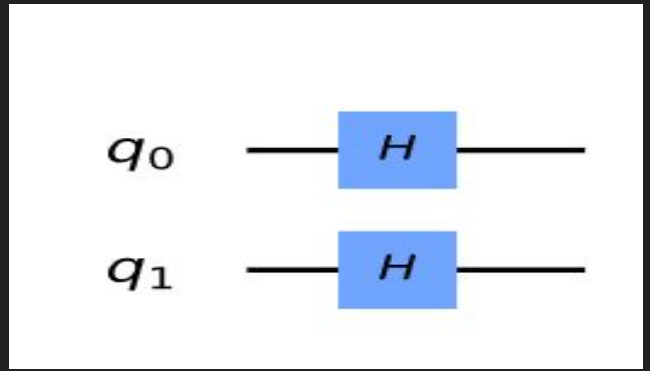
Quantum Approach:

Given n items each item has a 1 in 2^n chances of being the correct number. However using a combination of the quantum gates we have seen, we can create a superposition of every possible item in our list then raise the probability of the state's collapse into the number were looking for.

- Step 1: Create superposition of states:
  - Code:

```
n = 2
grover = QuantumCircuit(n)

for qubit in range(n):
    grover.h(qubit)
grover.draw('mpl')
```

  - Circuit:



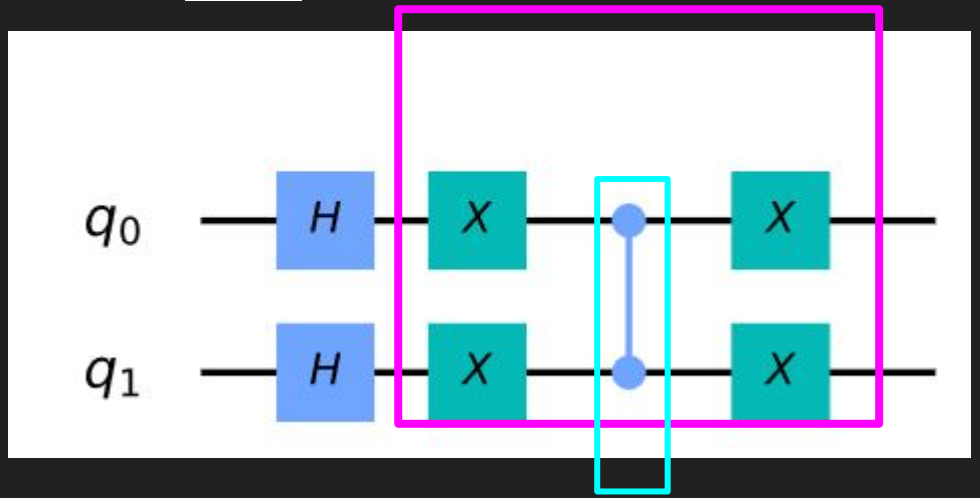
Grover's Algorithm:

- Step 2: Mark state |00⟩:
  - Code:

```
# apply oracle |w> = |00>
# manipulate the probabilty of superposition
# marking the state |00> with reflections
# manipulating probability.
for qubit in range(n):
    grover.x(qubit)

grover.cz(0, 1)

for qubit in range(n):
    grover.x(qubit)

grover.draw('mpl')
```

  - Circuit:

Oracle:



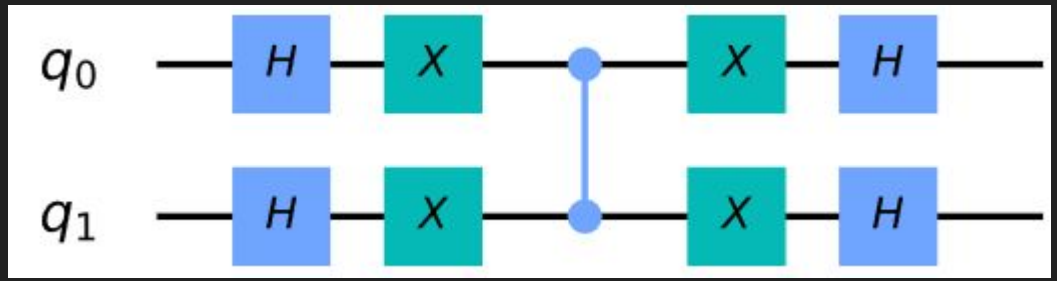
CZ-Gate, a combination z-gate and control gate

Grover’s Algorithm:

- Step 3: Apply Hadamard to output of oracle:
  - Code:

```
#Hadamard operations
for qubit in range(n):
    grover.h(qubit)
grover.draw('mpl')
```

  - Circuit:

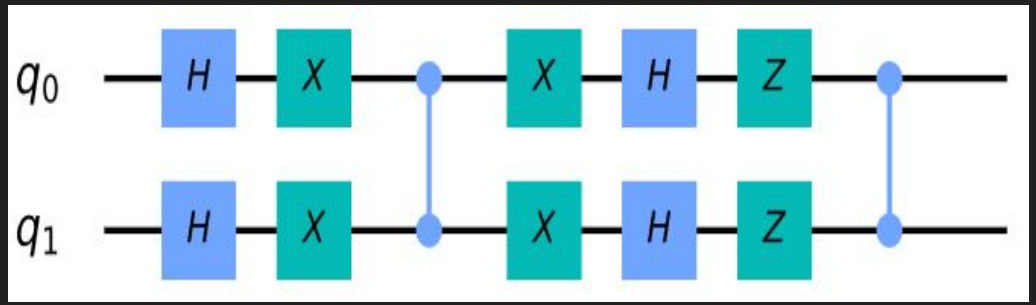


Grover’s Algorithm:

- Step 4: Apply reflections to every state other than |00⟩, reducing their probability:
  - Code:

```
# U reflections
for qubit in range(n):
    grover.z(qubit)
grover.cz(0,1)

grover.draw('mpl')
```

  - Circuit:

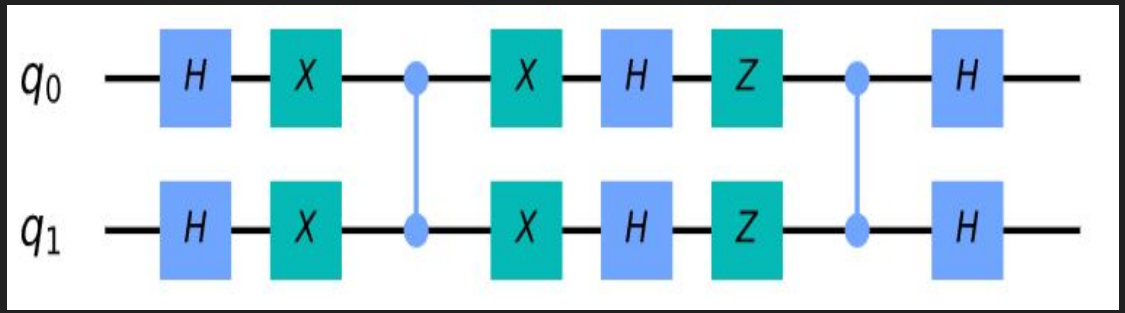


Grover’s Algorithm:

- Step 5: Apply final Hadamard gate:
  - Code:

```
#hadamard on last two qubits
for qubit in range(n):
    grover.h(qubit)


grover.draw('mpl')
```

  - Circuit:



Grover's Algorithm:

- Output: Find least busy Quantum device and execute circuit:
  - Code:

```
device = least_busy(provider.backends(simulator=False))
print("Running on current least busy device: ", device)
```

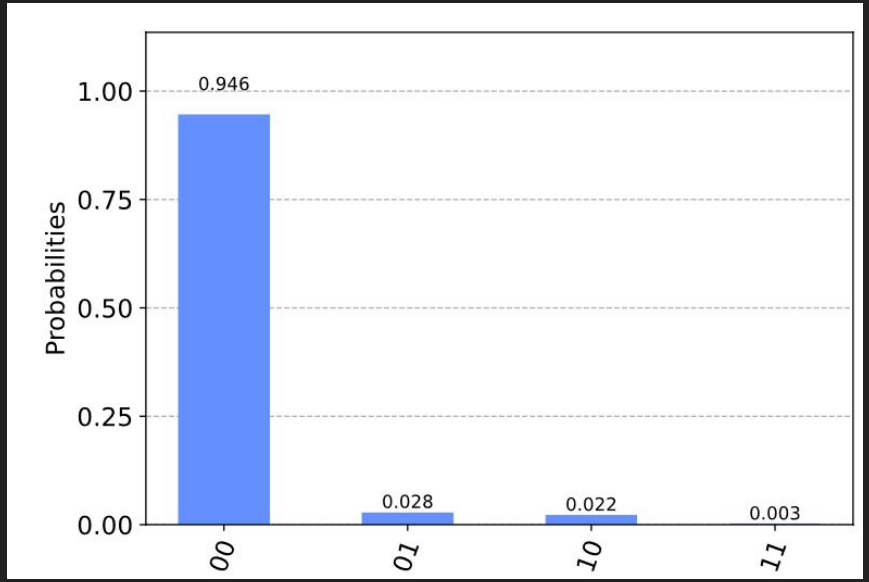
```
Running on current least busy device:  ibmq_ourense
```

```
# Run our circuit on the least busy backend. Monitor the execution of the job in the queue
from qiskit.tools.monitor import job_monitor
job = execute(grover, backend=device, shots=1024, max_credits=10)
job_monitor(job, interval = 2)
```

```
Job Status: job has successfully run
```

```
# Get the results from the computation
results = job.result()
answer = results.get_counts(grover)
plot_histogram(answer)
```

  - Output:



Review:

- In this presentation we covered the basics of qubits and quantum gates through the use of Linear Algebra and Probability In Order to provide a more concrete understanding of qubits and quantum operations. We then used this knowledge to go ahead and implement Grover’s search algorithm on a real quantum computer.