CSCI 4140
Quantum Phase Estimation

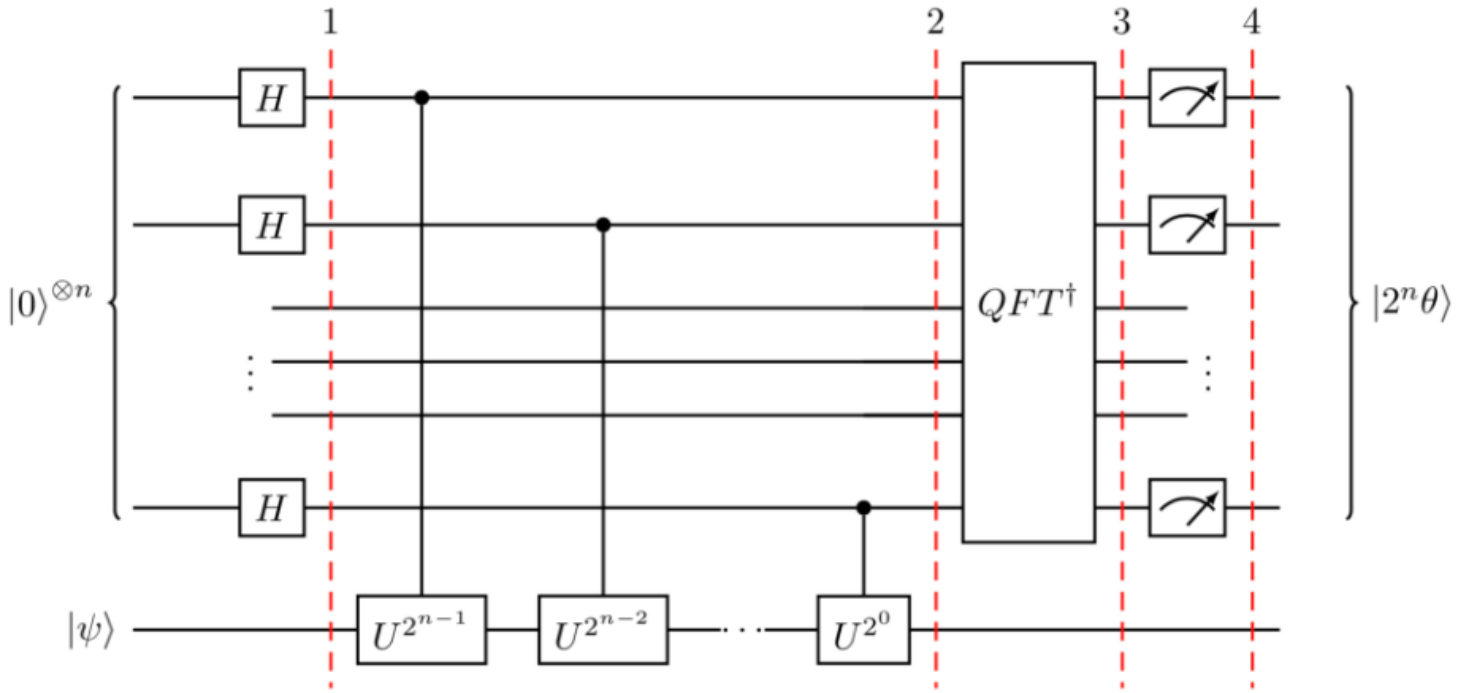
Mark Green

Faculty of Science

Ontario Tech

Introduction

- Implement quantum phase estimation as a component required for Shor's algorithm
- The algorithm is:



Introduction

- We've already implemented the inverse QFT and we know how to do Hadamard gates, but we need a candidate for U
- Select something we know the phase of, so we can test the algorithm, the T gate

$$T|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = e^{\frac{i\pi}{4}}|1\rangle$$

- In the case the phase is $\theta = {}^{1}/_{8}$ and the eigenvector is |1>

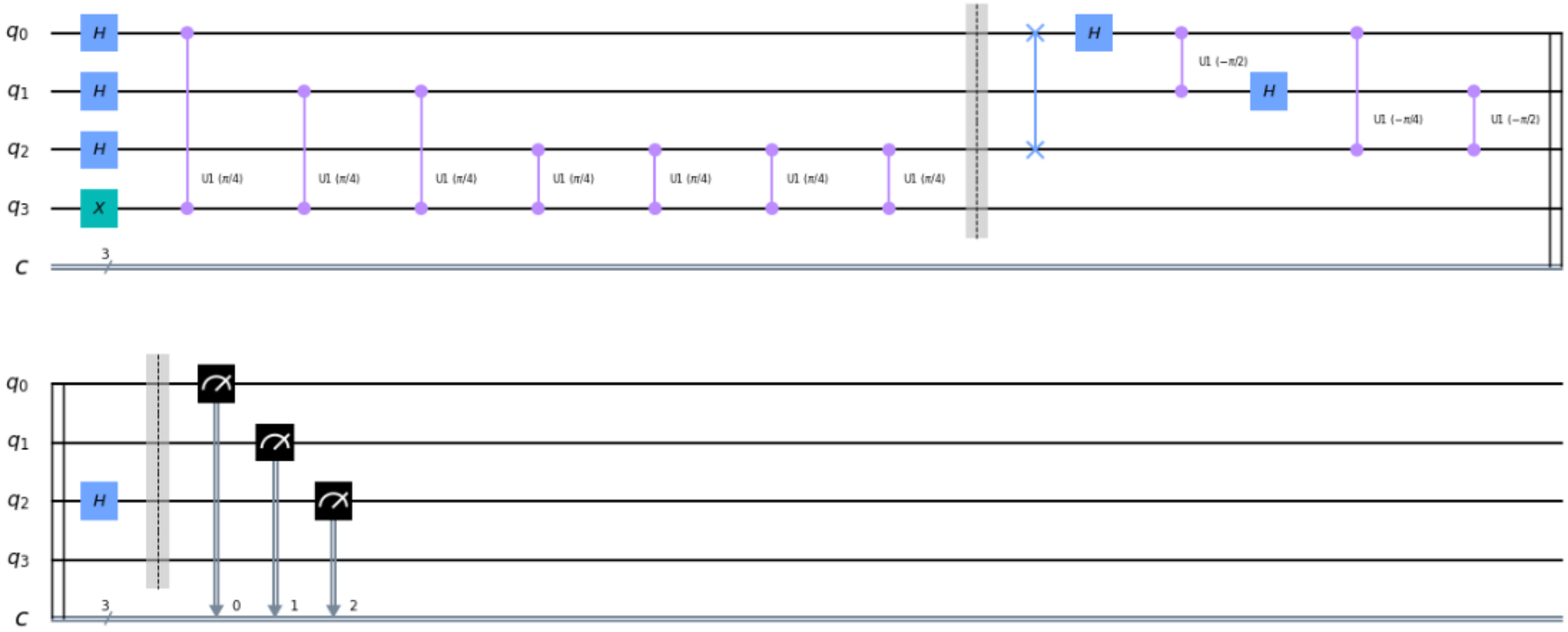
Implementation

- In this case we only need 3 qubits to accurately estimate the phase, and three classical bits for measuring it
- The eigenvector is |1>, so we will need a fourth qubit for it
- We will borrow our inverse QFT from the quantum Fourier transform lecture, since it’s a major part of the algorithm
- The Qiskit code for this is shown on the next slide, and the following slide has the circuit

Implementation

```
qpe = QuantumCircuit(4, 3)
# Add the eigenvector |1>
qpe.x(3)
# Add the Hadamard gates
for qubit in range(3):
    qpe.h(qubit)
# Add the powers of the controlled T gate
repetitions = 1
for counting_qubit in range(3):
    for i in range(repetitions):
        qpe.cu1(math.pi/4, counting_qubit, 3); # This is C-U
    repetitions *= 2
qpe.barrier()
# Apply inverse QFT
qft_dagger(qpe, 3)
# Measure
qpe.barrier()
for n in range(3):
    qpe.measure(n,n)
qpe.draw('mpl')
```

Implementation



Implementation

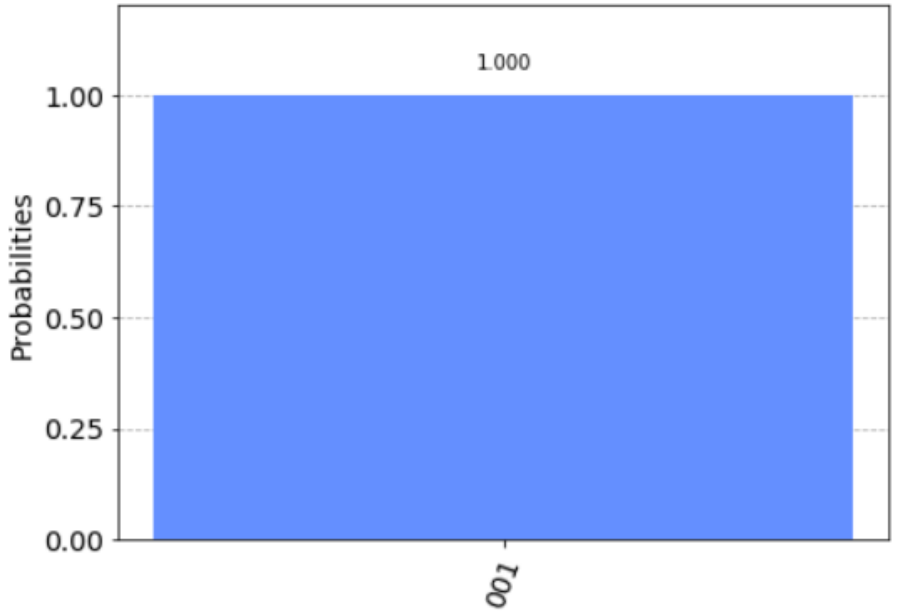
- The next slide shows what happens when we run the circuit
- The result that we get is 001
- Remember that this is θ multiplied by 2^n, in this case n=3
- This means that our result is:

$$\theta = \frac{1}{2^3} = \frac{1}{8}$$

- This is exactly the value that we were expecting, so in this case we get an accurate result

Implementation

```
backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpe, backend=backend, shots=shots).result()
answer = results.get_counts()

plot_histogram(answer)
```



Implementation

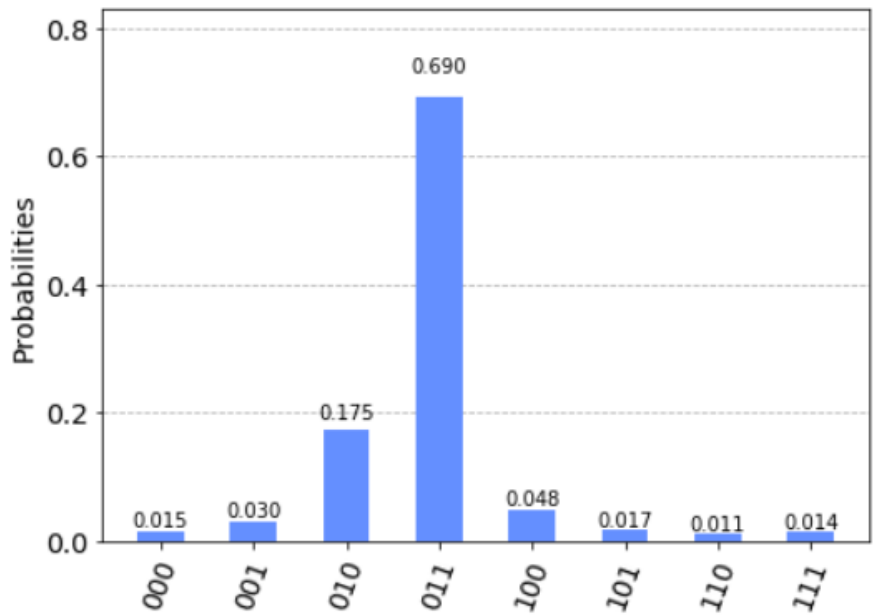
- What happens if the angle doesn't have an exact representation in n bits, that is, we can only produce an approximation
- The controlled U1 gate that we've used in our circuit can have any angle, we can try $\theta = {}^{1}/_{3}$, which doesn't have an exact representation as a binary fraction
- In this case we won't be able to produce the exact result
- The circuit for this is shown on the next slide
- The results are shown on the following slide

Implementation

```
# Create and set up circuit
qpe2 = QuantumCircuit(4, 3)

# Apply H-Gates to counting qubits:
for qubit in range(3):
    qpe2.h(qubit)

# Prepare our eigenstate |psi>:
qpe2.x(3)

# Do the controlled-U operations:
angle = 2*math.pi/3
repetitions = 1
for counting_qubit in range(3):
    for i in range(repetitions):
        qpe2.cu1(angle, counting_qubit, 3);
    repetitions *= 2

# Do the inverse QFT:
qft_dagger(qpe2, 3)

# Measure of course!
for n in range(3):
    qpe2.measure(n,n)
```

Implementation

```
backend = Aer.get_backend('qasm_simulator')
shots = 4096
results = execute(qpe2, backend=backend, shots=shots).result()
answer = results.get_counts()

plot_histogram(answer)
```



Implementation

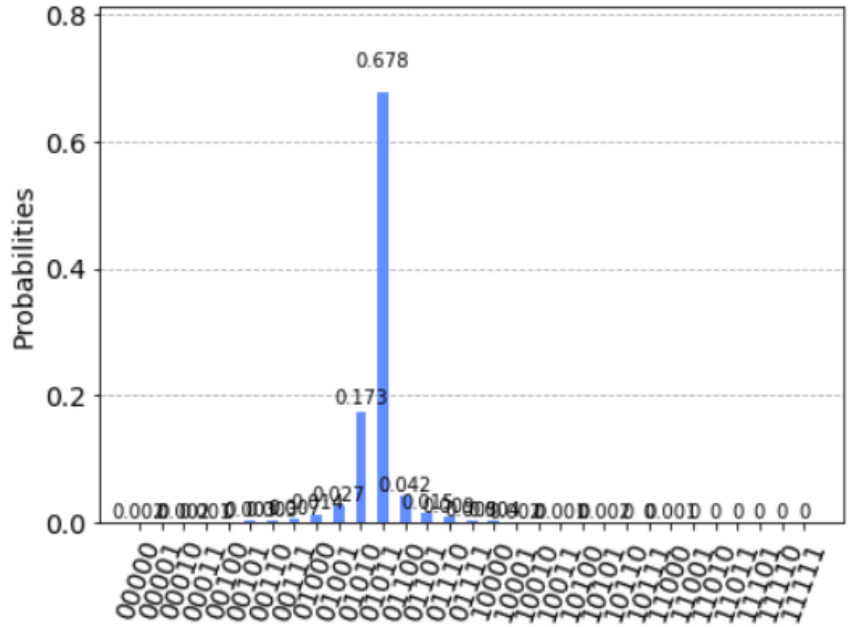
- The correct value is $\theta = 0.333333\ldots$
- We get the highest value at 011, which gives $\theta = {}^{3}/_{8} = 0.375$
- The next highest value is at 010, which gives $\theta = {}^{2}/_{8} = 0.25$
- The correct value is between them, but neither of them are particularly accurate
- To get more accurate results we need to increase the number of bits
- The slide shows the code for 5 bits, and the one after that the results of running it

Implementation

```
# Create and set up circuit
qpe3 = QuantumCircuit(6, 5)

# Apply H-Gates to counting qubits:
for qubit in range(5):
    qpe3.h(qubit)

# Prepare our eigenstate |psi>:
qpe3.x(5)

# Do the controlled-U operations:
angle = 2*math.pi/3
repetitions = 1
for counting_qubit in range(5):
    for i in range(repetitions):
        qpe3.cu1(angle, counting_qubit, 5);
    repetitions *= 2

# Do the inverse QFT:
qft_dagger(qpe3, 5)

# Measure of course!
qpe3.barrier()
for n in range(5):
    qpe3.measure(n,n)
```

Implementation

```
backend = Aer.get_backend('qasm_simulator')
shots = 4096
results = execute(qpe3, backend=backend, shots=shots).result()
answer = results.get_counts()

plot_histogram(answer)
```



Implementation

- The peak is at 01011=11, which is hard to read off of the graph
- This gives the estimate of $\theta = {}^{11}/_{32} = 0.344$
- The other large value is 01010=10
- This gives the estimate of $\theta = {}^{10}/_{32} = 0.313$
- These are much closer estimates
- We could try doing a simple average or a weighted average to get a closer result
- The weighted average gives 0.337698002350176 3

Summary

- Explored the implementation of quantum phase estimation
- Shown that it can only approximate some results if they don’t have an exact binary representation
- Accuracy can be improved by adding more qubits