Quantum Computing for Engineers

Osama Muhammad Raisuddin

July 13, 2023

Contents

This is the Abstract

1 Introduction

Quantum computing is an emerging area of research with great potential for engineering and scientific computing applications. Endowed with quantum mechanical properties such as superposition, entanglement and tunneling, quantum computers can enable exponential speedups over classical computers for some problems. The advantage can be realized in both time and energy consumption.

All classical computation can be performed on quantum computers using reversible versions of logic gates. However, this approach is inefficient and required large overheads. To take advantage of quantum computers new algorithms have been developed and are an active area of research. Due to stark differences with classical computation, the engineering community has limited exposure to quantum computing and quantum algorithms.

In this review we aim to provide an introduction to quantum computing with demonstration of concepts and ideas with code. Quantum algorithms relevant to engineers are presented in detail with implementation.

2 Quantum Information

2.1 Quantum States

2.2 Operators

2.3 Measurements

2.4 Composite Systems

2.5 Density Operators

2.6 Born Rule

This section should give an introduction to the state spaces of the qubits, operators, the Kronecker products, and densoty operator representation.

The Born Rule needs to be introduced. The no-cloning theorem can also be introduced here.

2.7 Postulates of Quantum Mechanics

2.7.1 State Space

2.7.2 Evolution

2.7.3 Measurement

2.7.4 Composite Systems

3 Quantum Computation

In this section we abstract away details of quantum information and the underlying physics of a quantum computer and present the computational model for quantum computing. Since all gate-based quantum computers share the same computational model, this is a safe starting point for beginners. Implementation of quantum algorithms does not require knowledge beyond these fundamental elements.

3.1 Qubits and Qudits

A 'qubit', or a quantum bit, is the quantum analogue of a classical bit. It is the elementary information unit in quantum computing. A qubit is a two-dimensional quantum system with dimensions typically labelled '0' and'1' analogous to the states of a classical bit. A qubit is exists in the Hilbert space $\mathcal{H} : \mathbb{C}^2$ and is represented in bra-ket (or Dirac) notation as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1}$$

where $|\alpha|^2 + |\beta|^2 = 1$ according to the Born rule and $|\psi\rangle \in \mathbb{C}^2$ denotes the quantum state of the qubit. $|0\rangle$ and $|1\rangle$ are the basis states in the $0-1$ basis, corresponding to the basis vectors $|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}$ and $|1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$. α and β are the 'probability amplitudes' of the state $|\psi\rangle$, corresponding to a probabilities of measuring the qubit $|\psi\rangle$ in states $|0\rangle$ and $|1\rangle$ with probailitities $|\alpha|^2$ and $|\beta|^2$ respectively. Note that once a measurement is performed on qubit $|\psi\rangle$ to obtain a measured state of either $|0\rangle$ or $|1\rangle$, the state $|\psi\rangle$ is destroyed and further measurements (without any other subsequent operations) will repeatedly yield the same state that was measured.

The state $|\psi\rangle$ is a 'ket'. The corresponding 'bra' $\langle\psi|$ is the adjoint, or the complex transpose of the 'ket'. As an example, the basis states form the kets $\langle 0| = \begin{pmatrix}1 & 0\end{pmatrix}$ and $\langle 1| = \begin{pmatrix}0 & 1\end{pmatrix}$.

A qudit is higher-dimensional generalization of of a qubit, s.t. $|\psi\rangle \in \mathbb{C}^d$, $d > 2$.

3.2 Registers of Qubits

A register of qubits is a collection of n qubits, whose combined state is represented as the quantum system $|\psi\rangle \in \mathbb{C}^{2^n}$, corresponding to a tensor product of

the individual Hilbert spaces of the qubits. The combined state $|\psi\rangle$ of two individual qubits $|\psi_0\rangle = \alpha_1|0\rangle + \beta_1|1\rangle$ and $|\psi_1\rangle = \alpha_2|0\rangle + \beta_2|1\rangle$ may be represented in Dirac notation as a Kronecker product of the individual qubits with various equivalent notations:

$$
\begin{aligned}
|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = |\psi_1\rangle|\psi_2\rangle = |\psi_1\psi_2\rangle \\
&= (\alpha_1|0\rangle + \beta_1|1\rangle)(\alpha_2|0\rangle + \beta_2|1\rangle) \\
&= \alpha_1\alpha_2|0\rangle|0\rangle + \alpha_1\beta_2|0\rangle|1\rangle + \alpha_2\beta_1|1\rangle|0\rangle + \beta_1\beta_2|1\rangle|1\rangle \\
&= \alpha_1\alpha_2|0\rangle \otimes |0\rangle + \alpha_1\beta_2|0\rangle \otimes |1\rangle + \alpha_2\beta_1|1\rangle \otimes |0\rangle + \beta_1\beta_2|1\rangle \otimes |1\rangle \\
&= \alpha_1\alpha_2|00\rangle + \alpha_1\beta_2|01\rangle + \alpha_2\beta_1|10\rangle + \beta_1\beta_2|11\rangle \\
&= \alpha_1\alpha_2 \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} + \alpha_2\beta_1 \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} + \alpha_1\beta_2 \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} + \beta_1\beta_2 \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} \\
&= \alpha_1\alpha_2 \begin{pmatrix}1\\0\\0\\0\end{pmatrix} + \alpha_2\beta_1 \begin{pmatrix}0\\1\\0\\0\end{pmatrix} + \alpha_1\beta_2 \begin{pmatrix}0\\0\\1\\0\end{pmatrix} + \beta_1\beta_2 \begin{pmatrix}0\\0\\0\\1\end{pmatrix} = \begin{pmatrix}\alpha_1\alpha_2\\\alpha_2\beta_1\\\alpha_1\beta_2\\\beta_1\beta_2\end{pmatrix}. \quad (2)
\end{aligned}
$$

Additonal qubits will follow the same pattern, resulting in an exponentially large state space for the qubits. Note that the ordering of qubits is arbitrary, and a change in ordering shuffles the representation of the state corresponding the Kronecker product; typically one may choose the most conventient order for their application.

3.3 Gates

In the gate-based quantum computing model operations on qubits are represented as quantum gates, or simply gates. Some of the gates are analogous to classical gate operations. However, some quantum gates may not have a corresponding classical counterpart.

Gates can be conveniently represented as complex matrices, in line with the column vector representation of qubits. The most basic gates are single qubit gates represented as $\mathbb{C}^{2\times 2}$. Some important single-qubit gates are the Pauli gate set $\{I, X, Y, Z\}$ and the Hadamard gate H.

Gate can also act on multiple qubits, in which case they can be represented in $\mathbb{C}^{2^n \times 2^n}$. Multiple qubit gates typically arise as controlled versions of gates, of which $CNOT$ is a commonly used one. The $SWAP$ gate is another common gate which swaps the states between qubits.

All quantum gates are unitary operations, which ensures normalization of quantum states in line with the Born rule. Consequantly, all quantum gates are also reversible operations, with the reverse operation simply being the Hermitian transpose or conjugate transpose of the quantum gate.

Gates acting on qubits are typically represented as left multiplication. As an example, consider a register of three qubits in the state $|000\rangle$. Applying a

Hadamard gate to the first qubit and a Pauli X gate to the last qubit (from the left) is represented as:

$$(H \otimes I \otimes X)|000\rangle = (H \otimes I \otimes I)|001\rangle = \frac{1}{\sqrt{2}}(|001\rangle + |101\rangle) \tag{3}$$

As another example, consider a register of two qubits with a Hadamard gate applied to the first qubit and a CNOT gate applied to the second qubit, controlled by the first qubit:

$$(|00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 11| + |11\rangle\langle 10|)(H \otimes I)|00\rangle = |00\rangle + |11\rangle \tag{4}$$

Note how the order of operations progresses from right to left. It is common practice not to include the I gate when the qubits on which the operation is applied is implied.

Similar to classical Boolean logic, quantum gates can also form a universal gate set. This implies that any arbitrary quantum gate can be approximated using a universal gate set. The Solovay-Kitaev theorem is a central theorem in quantum computing which shows that the approximation error using a universal gate set scales as $O(\log^c(\frac{1}{\epsilon}))$ for a single-qubit gate where $c \approx 2$ and $O(m \log^c(\frac{m}{\epsilon}))$ for a a set of m $CNOT$s and single-qubit unitaries. These correspond to a polylogarithmic increase in the original number of gates.

Underlying hardware implementations can have a variety of gate sets. Algorithms are typically agnostic to the hardware implementation, since the gates are converted to the target hardware's gate set in a process called transpilation.

3.4 Measurement

A quantum state is defined using probability amplitudes. In order to read a state a series of measurements of the state need to be performed, and the measurement statistics will correspond to the probability amplitudes of the quantum state.

In gate-based quantum computing projective measurements are typically used and are represented by a 'meter' symbol. The measurement operation is a projection onto the $|0\rangle$ $|1\rangle$ basis. As an example, consider a two-qubit system in the state $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$, with a measurement being performed on the first qubit. If the first qubit is measured in the state $|0\rangle$, the final state is:

$$(|0\rangle\langle 0| \otimes I)(\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)) = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle) \tag{5}$$

Similarly, if the first qubit is measured in the state $|1\rangle$ the final state is:

$$(|1\rangle\langle 1| \otimes I)(\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)) = \frac{1}{\sqrt{2}}(|10\rangle + |11\rangle) \tag{6}$$

Measurements may be performed to either read out an entire quantum register or as a flag to indicate successful operation by measuring ancilla qubits.

Note that measurements will provide the squared moduli (amplitudes) of the complex numbers defining the probability amplitude of a quantum state. In order to recover additonal information, like the arguments (phases), a process known as quantum state tomography [ref] is performed. Quantum states also have an overall phase. The overall phase is not measurable; only relative phase between two states is measurable.

3.5 Circuits

Working with algebraic forms of algorithms can be unwieldy and difficult to visualize. Quantum circuits are a convenient representation of qubits and the operation sequence.

As an example, consider a register of three qubits $|ABC\rangle$, with an H gate applied to the A and an X gate applied to C, followed by a $CNOT$ gate controlled by the A applied to B, and finally measurements on all the qubits. These operations can conveniently be represented in circuit form as:

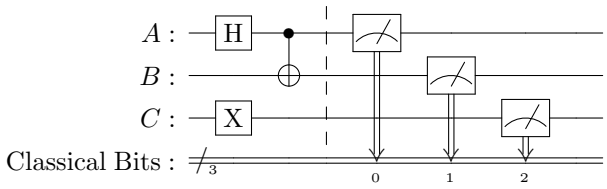

Figure 1: Circuit with gates and measurements.

Note that the operations are ordered from left to right, unlike operations ordered right to left in algebraic notation. A controlled operation can be applied conditioned on the control qubit being either in the state 1 or 0. This in indicated in a quantum circuit with a filled or empty circle respectively, as shown in Figure 2.

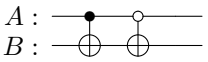

Figure 2: $CNOT$ gates conditioned on A in states 0 and 1.

Quantum registers can be ordered using either little-endian or big-endian notation. Little-endian ordering corresponds to the right-most qubit in algeraic notation indexed as 0 and big-endian corresponds to the left-most qubit indexed as 0. Qiskit uses little-endian ordering.

As an example, we repeat the example in Figure 1 but with $|ABC\rangle$ being denoted as $|\psi\rangle$. Consistent with Qiskit, in little-endian notation $|A\rangle$, $|B\rangle$, and $|C\rangle$ correspond to $|\psi_2\rangle$, $|\psi_1\rangle$, and $|\psi_0\rangle$. We provide an example code: starting with big-endian notation and converting to little-endian at the end.

```
#!/usr/bin/python3

from matplotlib import pyplot as plt
import qiskit

# Create a register of 3 qubits
myQRegister = qiskit.QuantumRegister(3, '\psi')

# Create a register of 3 classical bits
myCRegister = qiskit.ClassicalRegister(3,'Classical-Bits')

# Create a quantum circuit with using myRegister
myCircuit = qiskit.QuantumCircuit(myQRegister, myCRegister)

# Start working in big-endian order

# Add the gates in order of operation
myCircuit.h(0)
myCircuit.x(2)
myCircuit.cnot(0,1,ctrl_state='1')

# Insert a barrier to keep circuit organized
myCircuit.barrier()

# Measure all the qubits in myQRegister and store state in myCRegister
myCircuit.measure(myQRegister,myCRegister)

# Convert from big-endian to little-endian by reversing bits
myCircuit = myCircuit.reverse_bits()

# Draw the circuit
myCircuit.draw('mpl')
plt.show()
```

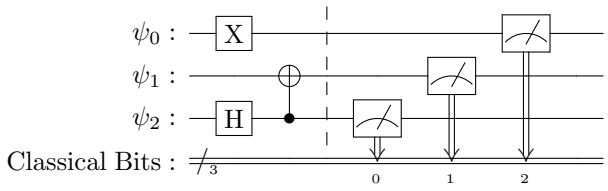

Figure 3: Output of Example code

3.6 Superposition

Superposition is a central property of quantum computing. A quantum state can exist in a superposition of states. As an example, a single qubit $|\psi\rangle$ can exist in a superposition of $|0\rangle$ and $|1\rangle$. A register of n qubits can exist in an exponentially large superposition of $N = 2^n$ states. The Hadamard gate puts qubits in superposition. As an example, we provide the following code to put qubits in uniform superposition and measure their states:

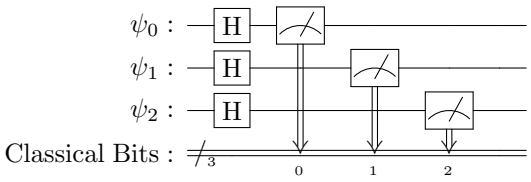

Figure 4: Circuit for uniform superposition.

```
#!/usr/bin/python3
from matplotlib import pyplot as plt
import qiskit
from qiskit import Aer
from qiskit.tools.visualization import plot_histogram

# Create a register of 3 qubits
myQRegister = qiskit.QuantumRegister(3, '\psi')

# Create a register of 3 classical bits
myCRegister = qiskit.ClassicalRegister(3,'Classical-Bits')

# Create a quantum circuit with using myRegister
myCircuit = qiskit.QuantumCircuit(myQRegister, myCRegister)

# Hadamard gates on al qubits
myCircuit.h(myQRegister)

# Measure all the qubits in myQRegister and store state in myCRegister
myCircuit.measure(myQRegister,myCRegister)


# Simulate the circuit
mySimulator = Aer.get_backend('aer_simulator')
result = mySimulator.run(myCircuit,shots=2**15).result()

# Plot a bar chart of all the results
plot_histogram(result.get_counts(), title='Uniform-Superposition')
```

```
plt.show()
```

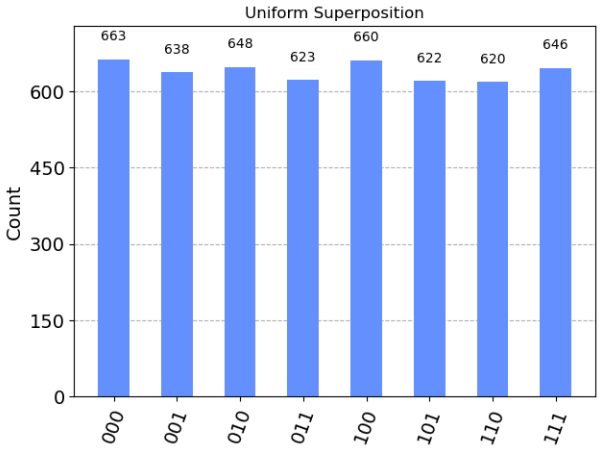

Figure 5: Output of Uniform Superposition code

3.7 Entanglement

Entanglement is another property unique to quantum computation. Entangled qubits have a correlated state. A maximal entanglement for a pair of qubits means that the full state of the one qubit can be determined completely by measuring the state of the second state. The simplest set of maximally entangled states are the Bell states:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), |\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle),$$
$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle), |\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle), \quad (7)$$

As an example, if the first qubit of the state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is measured as 0, the second qubit is also in the state 0, and if the first qubit is measured as 1, the second qubit is also in the state 1. Note that this is not the same as the state $|\psi\rangle = |00\rangle$, which is not an entangled state.

Mathematically, entangled qubits cannot be separated into a Kronecker product. The state $|00\rangle$ can be written as $|0\rangle \otimes |0\rangle$, however, it is impossible to separate $|\Psi^+\rangle$ into such a Kronecker product.

Here, we provide an example of a circuit creating the entangled state $|\Phi^+\rangle$:

```
#!/usr/bin/python3

from matplotlib import pyplot as plt
import qiskit
from qiskit import Aer
from qiskit.tools.visualization import plot_histogram

# Create a register of 2 qubits
myQRegister = qiskit.QuantumRegister(2, '\psi')

# Create a register of 2 classical bits
myCRegister = qiskit.ClassicalRegister(2,'Classical-Bits')

# Create a quantum circuit with using myRegister
myCircuit = qiskit.QuantumCircuit(myQRegister, myCRegister)

# Hadamard gates on first qubit
myCircuit.h(0)
# CNOT gate controlled by first qubit on second qubit
myCircuit.cnot(0,1)

# Measure all the qubits in myQRegister and store state in myCRegister
myCircuit.measure(myQRegister,myCRegister)


# Simulate the circuit
mySimulator = Aer.get_backend('aer_simulator')
result = mySimulator.run(myCircuit,shots=5120).result()

# Plot a bar chart of all the results
plot_histogram(result.get_counts(), title='Bell-State')

plt.show()
```

3.8 Reversible Computation

All quantum gates are reversible, since they are unitary operations. Therefore, the corresponding inverse operation of a gate O is $O^\dagger$, where $\dagger$ denotes the Hermitian conjugate of O.

Classical gates such as AND and OR are not reversible. However, a reversible version of these gates can be formed using ancilla qubits. Even though this approach does not make efficient use of quantum resources, it does imply that all classical computation can be performed on quantum computers.

Since measurements are projective operations, they are not reversible.

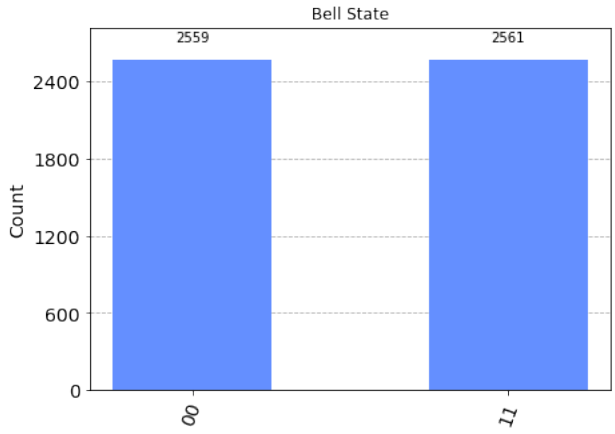

Figure 6: Output of Uniform Superposition code

3.9 Data Representation

Classical information is represented as bit strings in classical computers. This encoding scheme may be translated to quantum bits, and is known as basis embedding. However, this representation is not particularly compact since it does not make use of the additional available degrees of freedom in quantum states, namely the phase and probability amplitudes of the individual basis states.

Embedding data into the phase of the a basis state is known as phase or angle embedding. Similarly, embedding data into the probability amplitude of a basis state is known as amplitude encoding or amplitude embedding. Both amplitude embedding and phase embedding have the limitations of quantum states, namely the Born rule normalization and periodicity of phases respectively. Amplitude encoding is the method of choice for quantum algorithms related to scientific computing and engineering.

As an example, consider the vector $\mathbf{a} = \Sigma_{i=0}^{2^n-1} a_i \mathbf{e}_i$, where $\mathbf{e}_i$ denotes the i^{th} standard basis vector and n is the number of qubits. If the vector length is smaller than 2^n, the remainder of the vector may be zero padded without any loss of generality. An amplitude encoding of this vector will be $|\psi\rangle = \Sigma_{i=0}^{2^n-1} a_i |i\rangle / \|\mathbf{a}\|_2$, where $|i\rangle$ is the i^{th} basis vector of the quantum state.

3.10 Limitations

Quantum physics places some fundamental limits on possible operations using quantum computing. A well-known limitation is the no-cloning theorem, which state that an exact copy of a quantum state cannot be made. Note that imperfect copies are still possible to create, with known bounds on the error.

The normalization of a state and the periodicity of phase can also be considered limitations on the state.

All gate operations are unitary and linear. In order to apply nonlinear operations a projection onto a subspace of the overall linear space must be considered. One may visualize this as the shadow of a rotating stick under a collimated light source; rotations of the object correspond to unitary operations and the shadow under a collimated light source is a projection onto a plane, which does not respond linearly w.r.t. rotation angles. This property is exploited in various quantum computing algorithms, at the expense of ancilla qubits and non-zero probability of failure.

Amplifying the probability of a desirable quantum state amongst a superposition of undesirable states can also pose a challenging limitation. If the amplitude of the desired state is exponentially small, the probability of obtaining the desired state cannot be boosted without an exponential overhead. This is known as the post-selection problem.

Getting data in and out of a quantum register is also a challenging problem. I/O is expensive on quantum computers: preparing or reading out an arbitrary quantum state scales as $O(2^n)$.

Despite these limitations, there is potential for quantum computing to make an impact on scientific computation and engineering problems.

3.11 Access Models

Quantum computing algorithms make use of oracles. Oracles are useful when defining algorithms since the implementation of a matrix A might not be known or can be unclear. Quantum oracles are usually used to access the entries of a matrix. There are two access models commonly seen in quantum algorithms for matrix operations, the sparse acess model and a block encoding model.

A quantum oracle is a black box that contains the information $f(x)$ that encodes the problem that is to be solved. In the case of linear systems this would be the entries of the matrix.

There is an XOR oracle and a Phase oracle

There are a variety of different oracles depending on the problem. For the problems considered in this paper, the

In order to perform computations on quantum computers we need a method to access the problem information on a quantum device. For matrix operations this can entail access to the entries of the matrix A. An early access model, on which the HHL algorithm is based, is access to the matrix in the form of the unitary matrix operator $U = e^{itA}$, where A is a Hermitian matrix. In case the matrix A is not Hermitian, its Hermitian dilation $\begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix}$ can be used instead.

Block-encoding oracle is another access model which is of the form

$$U_A = \begin{pmatrix} A & * \\ * & * \end{pmatrix} \tag{8}$$

where $*$ are irrelevant entries.

This form is central to methods based on qubitization and quantum signal processing, which are the most powerful and optimal methods for most problems. The main idea is to apply the block-encoded unitary to a state

$$U_A|0\rangle|b\rangle = \begin{pmatrix} A & * \\ * & * \end{pmatrix} \begin{pmatrix} b \\ 0 \end{pmatrix} = \begin{pmatrix} Ab \\ * \end{pmatrix} = |0\rangle|Ab\rangle + |\perp\rangle \tag{9}$$

Measuring the first qubit in the state $|0\rangle$ indicates a successful matrix-vector multiplication. Note that although this 2×2 block encoding uses a single ancilla qubit, a register of qubits can also be used for block encoding which requires m ancilla qubits to be measured as $|0\rangle^{\otimes m}$.

The specifics of oracle implementations for access models are problem-dependent and beyond the scope of this review.

An example of a circuit using a sparse matrix access model to encode e^{itA} where A is a tridiagonal Laplacian matrix can be found in [Vasquez].

4 Quantum Algorithms for Engineering

4.1 Important Subroutines

4.1.1 Phase Kickback

Phase kickback is a uniquely quantum phnomenon where the control qubit in a controlled operation is affected by the operation, whereas qubit being controlled remains unchanged. Classically the control bits for logic gates remain unchanged.

Whenever a controlled gate is applied, the eigenvalues of the gate can be 'kicked back' to the phase of the control qubit.

Consider a unitary operation in its eigenbasis $U = \sum_i e^{i\lambda_i}|\lambda_i\rangle$. This operation controlled by one qubit can be written as

$$cU = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes \sum_i e^{i\lambda_i}|\lambda_i\rangle \tag{10}$$

where the first qubit is the control qubit. Now consider a quantum state $|\psi_i\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |\lambda_i\rangle$, where the first qubit is in unform superposition. Applying the controlled unitary operation cU on the state $|\psi_i\rangle$ gives:

$$\begin{aligned} cU|\psi_i\rangle &= (|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes \sum_i e^{i\lambda_i}|\lambda_i\rangle)(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |\lambda_i\rangle) \\ &= \frac{1}{\sqrt{2}}(|0\rangle|\lambda_i\rangle + e^{i\lambda_i}|1\rangle|\lambda_i\rangle) \\ &= \frac{1}{\sqrt{2}}(|0\rangle + e^{i\lambda_i}|1\rangle)|\lambda_i\rangle \end{aligned} \tag{11}$$

We note that the qubit register $|\lambda_i\rangle$ appears to be unchanged, while the control qubit has accumulated a phase of λ_i. This property is widely used in quantum subroutines like phase estimation.

4.1.2 Quantum Fourier Transform

The quantum Fourier transform is a discrete Fourier transform applied to probability amplitudes of a quantum state. This operation is exponentially faster on a quantum computer.

A discrete Fourier Transform takes as input $\psi \in \mathbb{C}^N$ and outputs $\phi \in \mathbb{C}^N$ s.t.

$$\phi_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \psi_j e^{2\pi i j k/N} \tag{12}$$

The quantum Fourier transform maps the basis states $|j\rangle$ of an input state $|\psi\rangle = \sum_{j=0}^{N-1} \psi_j |j\rangle$ to an output state $|\phi\rangle = \sum_{k=0}^{N-1} \phi_k |k\rangle$ as:

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k} |k\rangle \tag{13}$$

corresponding to the operation on an arbitrary state $|\psi\rangle$

$$\sum_{j=0}^{N-1} \psi_j |j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \sum_{j=0}^{N-1} \psi_j e^{2\pi i j k/N} |k\rangle \tag{14}$$

The probability amplitudes ϕ_k of $|\phi\rangle = \sum_{k=0}^{N-1} \phi_k |k\rangle$ are the discrete Fourier transform of the probability amplitues ψ_j of $|\psi\rangle = \sum_{j=0}^{N-1} \psi_j |j\rangle$.

Another convenient representation of the quantum Fourier transform is its product form. To do this, it is convenient to use the following labeling for the basis states instead:

For $N = 2^n$, where $n \in \mathbb{Z}^+$, the basis states $|j\rangle \in \{|0\rangle, ..., |2^n - 1\rangle\}$ may be relabeled using the binary notation of the integers j as follows:

$$j = j_1 2^{-1} + j_2 2^{n-2} + ... + j_n 2^0 :\rightarrow j_1 j_2 ... j_n \tag{15}$$

Similarly, the binary fraction is is denoted as:

$$j_l/2 + j_l + 1/4 + ... + j_m/2^{m-l+1} :\rightarrow 0.j_l j_{l+1} ... j_m \tag{16}$$

Using ths notation, the quantum Fourier transform on a basis state $|j_1 j_2 ... j_n\rangle$ may be rewritten as

$$|j_1 j_2 ... j_n\rangle \rightarrow \frac{(|0\rangle + e^{0.j_n}|1\rangle)(|0\rangle + e^{0.j_n j_{n-1}}|1\rangle)...(|0\rangle + e^{0.j_1 j_2 ... j_n}|1\rangle)}{2^{n/2}} \tag{17}$$

The Fourier transform is a unitary operation.

The ordering of the qubits is a common source of confusion in implementations. The sequence of SWAP gates at the end simply reverse the order of the qubits; this can be avoided by applying the gates

We provide as an example a Qiskit implementation of a quantum Fourier transform on a uniform superposition of basis states. A uniform superposition corresponds to $\phi_k = \frac{1}{\sqrt{N}} \forall k$, and its corresponding discrete Fourier transform is the delta function $\psi_j = \delta_{0,j}$.

```
#!/usr/bin/python3

from matplotlib import pyplot as plt
from qiskit import QuantumCircuit
from qiskit.circuit.library import QFT
from qiskit import Aer
from qiskit.tools.visualization import plot_histogram

beforeFT = QuantumCircuit(5)
# Initialize state in uniform superposition
beforeFT.h([0,1,2,3,4])

# Measure all qubits
beforeFT.measure_all()

# Simulate the circuit
mySimulator = Aer.get_backend('aer_simulator')
result = mySimulator.run(beforeFT, shots=2**20).result()

# Plot a bar chart of all the results
plot_histogram(result.get_counts(), bar_labels=False, title='Before QFT')

afterFT = QuantumCircuit(5)
# Initialize qubits
afterFT.h([0,1,2,3,4])

# Add Fourier transform operation
qft = QFT(num_qubits=5,do_swaps=False).to_gate()
afterFT.append(qft, qargs=[0,1,2,3,4])

# Measure all qubits
afterFT.measure_all()

# Decompose Fourier transform operation into gates for simulator
afterFT = afterFT.decompose(reps=2)

# Simulate the circuit
result = mySimulator.run(afterFT, shots=2**20).result()
```

```
# Plot a bar chart of all the results
plot_histogram(result.get_counts(), bar_labels=False, title='After QFT')

plt.show()
```

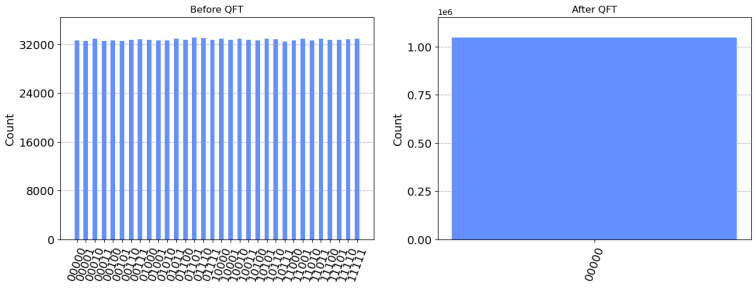

Figure 7: Basis states before and after a quantum Fourier transform

4.1.3 Phase Estimation

In simple terms, given an eigenstate λ_i of a unitary U, phase estimation provides an estimate of the corresponding eigenvalue $e^{i\lambda_i}$. Phase estimation uses the quantum Fourier transform to estimate the phase kicked back by a unitary operation. A series of controlled versions of the unitary operation are performed on an input state. Figure 8 shows an example with 4 control qubits. The control qubits are put in uniform superposition before applying the controlled operations. More specifically, j control qubits are used to apply the unitary U^{2^j} controlled by the j^{th} control qubit.

We know from phase kickbak that for $U = \sum_i e^{i\lambda_i}|\lambda_i\rangle$, with any input $|\psi\rangle = \sum_k c_k|\lambda_i\rangle$.

We know that the work register will remain unchanged when the controlled unitaries are applied, even if the input $|\psi\rangle$ is not an eigenstate of $U = \sum_i e^{i\lambda_i}|\lambda_i\rangle$, since we can decompose the input into the eigenbasis of U as $|\psi\rangle = \sum_k c_k|\lambda_k\rangle$.

Denoting $c_j U^{2^j}$ as the unitary U^{2^j} applied to the work register, controlled by the j^{th} qubit, form the phase kickback result we have

$$\prod_{j=0}^{J-1}(c_j U^{2^j})(\frac{1}{2^{\frac{J-1}{2}}}(|0\rangle+|1\rangle)^{\otimes J-1}|\psi\rangle)$$
$$= \frac{1}{2^{\frac{J-1}{2}}}\bigotimes_{j=0}^{J-1}(|0\rangle + e^{i\lambda_i 2^j}|1\rangle)|\psi\rangle$$
$$= \frac{1}{2^{\frac{J-1}{2}}}\sum_{k=0}^{2^{J-1}-1}(|0\rangle + e^{ik\lambda_i}|1\rangle)|\psi\rangle \tag{18}$$

Comparing *productformequation* with the product form of the quantum Fourier transform in 17, we can clearly see that performing the inverse quantum Fourier transform will give us

Using the phase kickbak result we have

clock and work register names

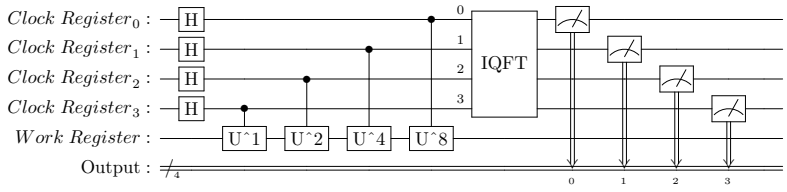

Figure 8: Circuit for quantum phase estimation

4.1.4 Amplitude Amplification

Amplitude Amplification is used to boost the success probability of a desired ouput. The procedure is proven to be optimal and provides a quadratic inprovement in success probability.

Consider an arbitrary quantum state $|\psi\rangle$ and a projective measurement

Consider a quantum algorithm $\mathcal{A}$ s.t.

$$\mathcal{A}|0\rangle^{\otimes n} = \sqrt{p}|\psi_{good}\rangle + \sqrt{1-p}|\psi_{bad}\rangle \tag{19}$$

where $|\psi_{good}\rangle$ is the desired output with success probability p. The amplitude amplification procedure will boost the success probability to $O(1)$ using $O(\sqrt{g})$ applicatoins of $\mathcal{A}$.

More specifically, this is achieved using $O(\sqrt{p})$ applications of the operator $\mathcal{Q}$:

$$\mathcal{Q} = \mathcal{A}\mathcal{S}_0\mathcal{A}^\dagger\mathcal{S}_{good} \tag{20}$$

where

$$\mathcal{S}_0 = \mathbb{I} - 2|0\rangle^{\otimes n}\langle 0|^{\otimes n} \tag{21}$$

is a reflection operation about the state $|0\rangle^{\otimes n}$ and

$$\mathcal{S}_0 = \mathbb{I} - 2|\psi_{good}\rangle\langle\psi_{good}| \tag{22}$$

is an oracle for reflection about the state $|\psi_{good}\rangle$. Intuitively these operations can also be interpreted as 'marking' the states $|0\rangle^{\otimes n}$. Note that the operations are Householder reflectors.

The overall circuit is provided in Figure 9

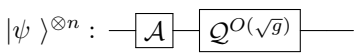

Figure 9: Circuit for amplitude amplification

Grover's search algorithm is a special case of amplitude amplification, which can search for an entry in an unstructured database of N entries in $O(\sqrt{N})$ time compared to $O(N)$ time needed classically.

4.1.5 Hamiltonian Simulation

Hamiltonian simulation is simply applying the operation $U \approx e^{-iHt}$ to a quantum state. Since H is a Hermitian matrix, U is unitary. The Hamiltonian simulation subroutine is important for simulating quantum systems like quantum chemistry problems and is also used in the HHL quantum linear system algorithm. It encodes the evolution of a quantum state according to the Schrodinger equation:

$$\frac{d}{dt}|\Psi(t)\rangle = -iH|\Psi(t)\rangle \tag{23}$$

where the Planck constant has been absorbed into the Hamiltonian H, which has the solution

$$|\Psi(t)\rangle = e^{-iHt}|\Psi(0)\rangle \tag{24}$$

Various methods exist for Hamiltonian simulation, the most important ones being Trotter-Suzuki product formulas, Taylor series approximations using linear combinations of unitaries, and quantum signal processing.

Suzuki-Trotter, or Trotter formulas use the Baker-Campbel-Hausdorff formula to approximate

$$e^{-i\mathbf{H}t} = e^{-i\sum_{j=0}^{k}\mathbf{H}_j t} = (e^{-i\sum_{j=0}^{k}\mathbf{H}_j t/r})^r \approx (\prod_{j=0}^{k} e^{-i\mathbf{H}_j t/r})^r + O(k^2t^2/r) \tag{25}$$

Higher order Trotter formulas can be constructed to further reduce the error. Although the error in Trotter formulas does not scale well, they have the advantage of not requiring additional (ancilla) qubits. As an example, the second order Trotter formula is

$$e^{-i\mathbf{H}t} = e^{-i\sum_{j=0}^{k}\mathbf{H}_j t} = (e^{-i\sum_{j=0}^{k}\mathbf{H}_j t/r})^r \approx (\prod_{j=0}^{k} e^{-i\mathbf{H}_j t/2r} \prod_{j=k}^{0} e^{-i\mathbf{H}_j t/2r})^r + O(k^3 t^3/r^2) \tag{26}$$

The Taylor series approximation using a linear combination of unitaries uses a slightly different approach by decomposing H as a sum of unitaries H_l

$$U = e^{-i\mathbf{H}t} = (e^{-i\mathbf{H}t/r})^r = (U_r)^r \tag{27}$$

$$U_r \approx \hat{U}_r = \sum_{k=0}^{K} \sum_{l_1,\ldots,l_k=1}^{L} \frac{(-it/r)^k}{k!} \alpha_{l_1} \ldots \alpha_{l_k} \mathbf{H}_{l_1} \ldots \mathbf{H}_{l_k} \tag{28}$$

A choice of $K = O(\frac{\log(r/\epsilon)}{\log\log(r/\epsilon)})$ leads to a precision $\|U - (\hat{U}_r)^r\|_2 \leq \epsilon$, an exponential improvement over Trotter simulations. However, to apply the sum in Equation 28 the linear combination of unitaries method is used, which requires ancilla qubits and has a non-zero probability of success.

The quantum signal processing method uses a complex polynomial approximation of $e^{-i\mathbf{H}t}$ using the quantum signal processing protocol. This involves applying the quantum signal processing circuit

$$U_\Phi(H) = e^{-i\phi_0 Z} \prod_{j=1}^{d} [O(H) e^{-i\phi_j Z}] \tag{29}$$

where ϕ_j are quantum signal processing angles independent of the particular problem and $O(H)$ is a block encoding of a Hamiltonian. The error decreases exponentially with the number of phase angles used. The phase angles may be calculated using QSPPACK. This method is also referred to as Qubitization.

We provide here as an example a Hamiltonian simulation of $\mathbf{H} = X + Z$ using the first- and second- order Trotter method for for various t, for varying numbers of Trotter steps, with the error plots shown in 4.1.5:

```
#!/usr/bin/python3

import qiskit
from qiskit import Aer
from scipy.linalg import expm
import numpy as np
import matplotlib.pyplot as plt

min_t = 1
```

```
max_t = 5

# Number of Trotter Steps
m = [10,20,40,80,100, 200, 400, 800, 1000, 2000, 4000, 8000, 10000]

# 1st order Trotter
for t in range(min_t,max_t+1):
    # Calculate exact solution classically
    exact_solution = expm( -t * 1j * ( np.array([[0, 1], [1, 0]])
      + np.array([[1, 0], [0, -1]]) ) )

    errors = []
    for r in m:
        # Create a register of 1 qubit
        myQRegister = qiskit.QuantumRegister(1, '\psi')

        # Create a quantum circuit with using myRegister
        myCircuit = qiskit.QuantumCircuit(myQRegister)
        for _r in range(r):
            myCircuit.rx(t*2/r,0)
            myCircuit.rz(t*2/r,0)

        # Simulate the circuit to obtain overall unitary of Trotterization
        mySimulator = Aer.get_backend('unitary_simulator')
        result = mySimulator.run(myCircuit).result()
        finalUnitary = result.get_unitary()

        # Compare circuit unitary with exact unitary
        errors.append( np.linalg.norm(finalUnitary - exact_solution,2) )

    plt.loglog(m,errors)

# 2nd order Trotter
for t in range(min_t,max_t+1):
    # Calculate exact solution classically
    exact_solution = expm( -t * 1j * ( np.array([[0, 1], [1, 0]])
      + np.array([[1, 0], [0, -1]]) ) )

    errors = []
    for r in m:
        # Create a register of 1 qubit
        myQRegister = qiskit.QuantumRegister(1, '\psi')

        # Create a quantum circuit with using myRegister
        myCircuit = qiskit.QuantumCircuit(myQRegister)
        for _r in range(r):
```

```
            myCircuit.rx(t/r,0)
            myCircuit.rz(t*2/r,0)
            myCircuit.rx(t/r,0)

        # Simulate the circuit to obtain overall unitary of Trotterization
        mySimulator = Aer.get_backend('unitary_simulator')
        result = mySimulator.run(myCircuit).result()
        finalUnitary = result.get_unitary()

        # Compare circuit unitary with exact unitary
        errors.append( np.linalg.norm(finalUnitary - exact_solution,2) )

    plt.loglog(m, errors, linestyle='dashed')

plt.xlabel('# of Trotter steps')
plt.ylabel('|error|_2')
plt.legend(['t = {}, 1st Order'.format(t) for t in range(min_t, max_t+1)]
        +['t = {}, 2nd Order'.format(t) for t in range(min_t, max_t+1)])
```

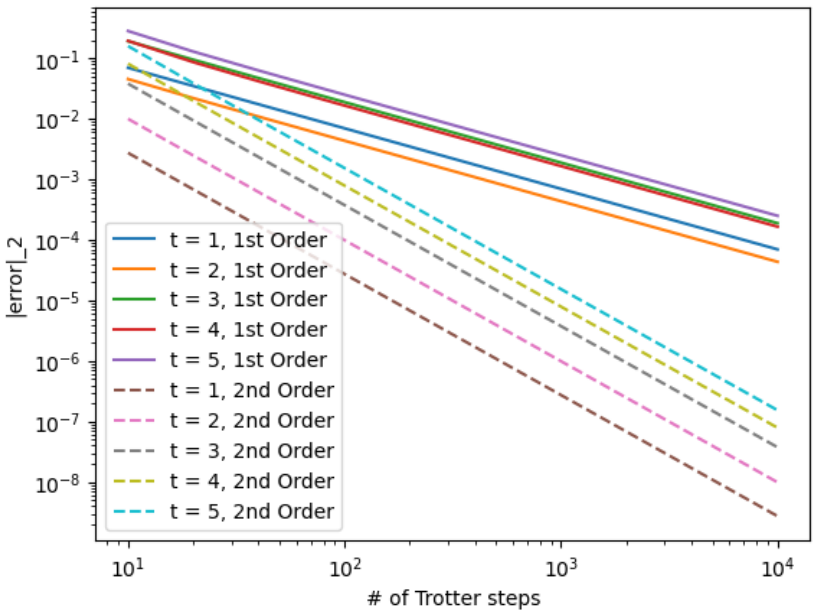

Figure 10: Output of Trotter Code

4.2 Quantum Linear System Algorithms

Quantum linear system algorithms are perhaps the most lucrative area of quantum computing for scientific and engineering computation.

The quantum linear system algorithms solve the quantum linear system problem. The quantum linear system problem is slightly different from the classical linear system problem and can formally be stated as

Given a quantum state $|\mathbf{b}\rangle$, prepare a state $|\mathbf{x}\rangle$ with precision ϵ s.t. $\mathbf{Ax} = \mathbf{b}$.

Due to normalization of quantum states, the quantum state $|\mathbf{x}\rangle$ is proportional to the solution $\mathbf{x}$. Furthermore, unlike classical solutions, the quantum state encodes the solution instead of the entire solution being available classically.

A plethora of quantum linear system algorithms exist, of which the most notable are the HHL algorithm, the LCU algorithm, and the QSP algorithm.

The HHL algorithm is the first proposed algorithm and it uses Hamiltonian simulation to apply controlled versions of e^{-iAt} on the state $|b\rangle$. Since A and e^{-iAt} share the same eigenvectors, the eigenvalues of e^{-iAt} are kicked back to the control qubits. Using the quantum Fourier transform, eigenvalues are encoded in the amplitudes of the control qubits. Finally, a controlled rotation on an ancilla is used to invert the eigenvalues. The computation is then reversed to disentangle the registers, and the ancilla is measured. Measurement of the ancilla in the desired state indicates successful solution.

The LCU algorithm works by using a Chebyshev polynmial approximation $P(x) \approx 1/x$ over the interval $[-1, -1/\kappa] \cup [1/\kappa, 1]$, where κ is the condition number of the system. The matrix polynomial approximation $P(A)$ is applied to the state $|b\rangle$ to obtain the approximate solution $A^{-1}b \approx P(A)b$. However, applicatoin of a linear combination of unitaries has an additional overhead in the form of ancilla qubits.

The QSP method circumvents the issue of ancilla qubits by applying $P(A)$ as a product of unitary operations rather than a sum. This entails finding a sequence of angles ϕ corresponding to the desired polynomial and a block encoding of the matrix.

4.3 Quantum Algorithms for Systems of Ordinary Differential Equations

Several algorithms have been proposed for linear, nonlinear, homogeneous, and non-homogeneous systems of ordinary differential equations. For nonlinear systems algorithms using Koopman and Carleman linearization have been proposed. Exponential speedups have theorized for the linear systems of ordinary differential equations. Since the non-homogeneous system of ordinary differential equations is of general interest, we provide here an overview of the algorithm.

We also note that higher order systems of ordinary differential equations can easily be converted to linear systems, therefore this algorithm holds for the general case of linear ordinary differential equations.

The algorithm involves solving a large linear system that encodes the evolution of an ordinary differential equation at all time steps, intermediate and final. The encoding is performed using a Taylor series approximation of the general matrix exponential, which propagates the solution in time.

4.4 Quantum Algorithms for Partial Differential Equations