Analysis of the Harrow–Hassidim–Lloyd (HHL) Algorithm Through Comparison with Classical Linear System Solvers

Muhammad Fatih Irkham Mauludi - 13524004
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13524004@std.stei.itb.ac.id, mhmd.fatih.im@gmail.com

***Abstract*—Quantum Computing Technology has revolutionized the world in many way. The emergenceo of many quantum algorithm must also be studied thoroughly. One of the proposed algorithm is the Harrow-Hassidim-Lloyd algorithm. In this study we will conduct an analysis of the HHL algorithm and compare it with mainstream existing classical linear system solver.**

***Index Terms*—Quantum Computing, Harrow-Hassidim-Lloyd, Linear System Problem**

I. Introduction

Quantum Computing Technology has revolutionized the world in many way. The emergenceo of many quantum algorithm must also be studied thoroughly. One of the proposed algorithm is the Harrow-Hassidim-Lloyd algorithm. In this study we will conduct an analysis of the HHL algorithm and compare it with mainstream existing classical linear system solver.

II. Literature Review

A. Linear System Problem (LSP)

A System of Linear Equations (SPL) is a finite set of linear equations with the same variables. The general form of SPL with m equations and n variables is

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n & = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n & = b_2 \\ \cdots & \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n & = b_m \end{cases}$$

or, dalam in form of matrix multiplication and augmented matrix,

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} & b_m \end{array}\right]$$

The solution from SPL can be a single solution, many solutions, or no solution. Commonly used solution methods include Gauss elimination, Gauss-Jordan elimination, Cramer's rule, and the inverse matrix method.

B. Gauss and Gauss-Jordan Elimination

In the Gaussian elimination method, the SPL is first converted into a *augmented* matrix with the coefficient matrix x_i on the left and the constant b_i on the right. Then, OBE is run on the *augmented* matrix to form a row echelon matrix to the left of the *augmented* matrix.
Finally, the equation in the matrix is returned to its initial equation form and then solved using the backward substitution technique (*backwards substitution*).

$$\begin{cases} x_1 + a'_{12}x_2 + a'_{13}x_3 + \cdots + a'_{1n}x_n & = b'_1 \\ x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n & = b'_2 \\ \cdots & \\ x_n & = b'_m \end{cases}$$

Furthermore, in the Gauss-Jordan elimination method, the SPL is also converted into *augmented* matrix form. However, in this method, the *augmented* matrix is converted using OBE into a reduced echelon matrix. From the reduced row echelon matrix the value of x_i can be directly determined.

C. Tensor Product

The tensor product is a bilinear operation that combines two vector spaces into a single vector space whose dimension is the product of the original dimensions.

Let V and W be vector spaces over a field $\mathbb{F}$. Their tensor product, denoted by $V \otimes W$, is a vector space together with a bilinear map,

$$\otimes : V \times W \to V \otimes W,$$

which maps the pair $(v, w), \quad v \in V, \quad w \in W$ to an element of vector space $V \otimes W$ denoted as $v \otimes w$.

If $V = \mathbb{F}^m$ and $W = \mathbb{F}^n$, then

$$\mathbf{v} \otimes \mathbf{w} = \begin{pmatrix} v_1 \otimes \mathbf{w} \\ v_2 \otimes \mathbf{w} \\ \vdots \\ v_m \otimes \mathbf{w} \end{pmatrix} \in \mathbb{F}^{mn},$$

For example if $\mathbf{v} = (1, 2, 3)^T \in \mathbb{R}^3$ and $\mathbf{w} = (4, 5, 6, 7)^T \in \mathbb{R}^4$ then,

$$\mathbf{v} \otimes \mathbf{w} = \begin{pmatrix} 1 \otimes (4,5,6,7)^T \\ 2 \otimes (4,5,6,7)^T \\ 3 \otimes (4,5,6,7)^T \end{pmatrix} = \begin{pmatrix} 1 \cdot 4 \\ 1 \cdot 5 \\ 1 \cdot 6 \\ 1 \cdot 7 \\ 2 \cdot 4 \\ 2 \cdot 5 \\ 2 \cdot 6 \\ 2 \cdot 7 \\ 3 \cdot 4 \\ 3 \cdot 5 \\ 3 \cdot 6 \\ 3 \cdot 7 \end{pmatrix} = \begin{pmatrix} 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 10 \\ 12 \\ 14 \\ 12 \\ 15 \\ 18 \\ 21 \end{pmatrix} \in \mathbb{R}^{12}$$

The operation of doing a tensor product n times to a vector space V with itself can be denoted as,

$$V^{\otimes n} = \underbrace{V \otimes V \otimes \cdots \otimes V}_{n \text{ times}}$$

D. Quantum Computation

Quantum Computing (QC) is a computing paradigm which utilize the quantum nature of the universe to provide faster computation time that would usually be impossible to be implemented in classical computing. Recall that classical computing is a paradigm which uses standard bit as its mathematical model which can have a 0/1 state [1].

The study of quantum computing can be classified into two major field, first one is the study of construction of the quantum computer itself using known quantum property of the universe (*e.g.* quantum entanglement, superposition, etc), and second is the study of quantum algorithms using a mathematical model known as quantum circuits. In this paper, we will not discuss the physics related to quantum computer and only focus on the latter field of study.

E. Qubits

A quantum computer uses qubits to represent units of information. Each qubit takes a value which is in a quantum superposition between 0 (False) and 1 (True). We can represent this superposition state using vector in the $\mathbb{C}^2$ vector space,

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

Where,

$$\alpha, \beta \in \mathbb{C}$$

$$|\alpha|^2 + |\beta|^2 = 1$$

The complex vector space is used so we can represent the quantum phase of a qubit (See section II-F on Quantum Phase). However, the standard way to represent a qubit state is to use the Dirac bra-ket notation and we usually denote the state using the Greek letter ψ (psi).

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha\,|0\rangle + \beta\,|1\rangle$$

Where $|0\rangle$ and $|1\rangle$ represent the standard basis vector in $\mathbb{C}^2$,

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix},\ |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

We will also define the following convention regarding tensor product in bra-ket notation,

$$|a\rangle \otimes |b\rangle = |a\rangle\,|b\rangle = |ab\rangle$$

$$|a\rangle^{\otimes n} = \underbrace{|aaa\ldots a\rangle}_{n \text{ times}} = \underbrace{|a\rangle \otimes |a\rangle \otimes \cdots \otimes |a\rangle}_{n \text{ times}}$$

A quantum register of size n is a collection of n qubits which holds information encoded as binary form. To represent the state of a quantum register, we use the tensor product of all the qubits in that register,

$$|\psi_1\psi_2\ldots\psi_n\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$$

For example, if we have a 4-qubit register labelled as register r and the state of each qubit of the register are,

$$\begin{aligned} |\psi_1\rangle &= |1\rangle \\ |\psi_2\rangle &= |0\rangle \\ |\psi_3\rangle &= |0\rangle \\ |\psi_4\rangle &= \frac{1}{2}\,|0\rangle + \frac{\sqrt{3}}{2}\,|1\rangle \end{aligned}$$

Then the state of the register r can be represented as follow,

$$\begin{aligned} |r\rangle &= |1\rangle \otimes |0\rangle \otimes |0\rangle \otimes \left(\frac{1}{2}\,|0\rangle + \frac{\sqrt{3}}{2}\,|1\rangle\right) \\ &= \frac{1}{2}\,|1000\rangle + \frac{\sqrt{3}}{2}\,|1001\rangle \end{aligned} \tag{1}$$

Suppose that the register holds the information for a decimal number encoded in binary form, since 1000_2 = 8 and 1001_2 = 9 then we can say that register r is in a superposition state where it simultaneously holds the number 8 and 9.

After a quantum computation, a qubit will usually be measured. When measured, the superposition state of that qubit will collapse to the measured state with probability $|\alpha|^2$ to be measured as 0 and $|\beta|^2$ to be measured as 1. From the previous example at (1), we can say that the register has a $|\frac{1}{2}|^2 = 25\%$ chance to be measured as 8 and $|\frac{\sqrt{3}}{2}|^2 = 75\%$ chance to be measured as 9.

F. Quantum Phase

In quantum computation, the qubits holds an attribute of phase which makes it differ from classical computation. A phase can be represented as a rotation of the state vector using the Euler's formula,

$$e^{i\theta} = \cos\theta + i\sin\theta \tag{2}$$

In particular, it is important to distinguish between *global phase* and *relative phase*.

1) ***Global Phase****:* A *global phase* is an overall phase factor multiplying the entire quantum state.

Two qubit state that differ only by a global phase represent the same physical state, because the global phase applied to the state will have no observable physical effect. In other words, two qubit in which its state only differ in global phase will have the same measurement probability.

For example if we have the qubit state,

$$|\psi_1\rangle = \alpha|0\rangle + \beta|1\rangle,$$

Then for any $\theta \in \mathbb{R}$,

$$|\psi_2\rangle = e^{i\theta}|\psi_1\rangle$$

We can say that $|\psi_1\rangle$ and any $|\psi_2\rangle$ will always have the same measurement probability.

2) ***Relative Phase****:* A *relative phase* is a phase difference between components of a superposition. Unlike global phase, relative phase *is physically observable* and can affect measurement outcomes. Relative phase will differ if only one component (either the $|0\rangle$ or $|1\rangle$) of the state vector is rotated (i.e. multiplied by $e^{i\theta}$).

For example consider these two qubit states,

$$\begin{aligned}|\psi_1\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\\ |\psi_2\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + e^{i\pi}|1\rangle)\\ &= \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\end{aligned}$$

We can see that both $|\alpha|^2$ and $|\beta|^2$ component is equal between state $|\psi_1\rangle$ and $|\psi_2\rangle$, therefore both states have equal measurement probability. However, they differ in relative phase by π. Regarding relative phase difference, we define four more basis state which can be used to represent a qubit state,

$$\begin{aligned}|+\rangle &= \frac{\sqrt{2}}{2}(|0\rangle + |1\rangle)\\ |-\rangle &= \frac{\sqrt{2}}{2}(|0\rangle - |1\rangle)\\ |i\rangle &= \frac{\sqrt{2}}{2}(|0\rangle + i|1\rangle)\\ |-i\rangle &= \frac{\sqrt{2}}{2}(|0\rangle - i|1\rangle)\end{aligned} \tag{3}$$

We will see more about the implication of relative phase difference in the section II-G about quantum logic gates.

G. Quantum Logic Gates

Quantum logic gates are unitary matrix operators acting on one or more qubits which transform the superposition state from one state to another while still preserving the norm of the qubit state.

Recall that a qubit state is just a vector, therefore a transformation operation on that vector can always be represented as a left-multiplication by a unitary matrix. Mathematically, a quantum gate U satisfies

$$U^\dagger U = I$$

The following is some of the most used general quantum logic gates,

1) Identity Gate:

$|q\rangle$ — $\boxed{I}$ —

$$I = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}, \qquad I|\psi\rangle = |\psi\rangle.$$

The identity gate leaves the state unchanged.

2) Pauli-X Gate (NOT Gate): The Pauli-X Gates performs a π radian rotation of a qubit state around the X-axis. Therefore it is equivalent with "negating" the qubit, and usually this gate can be thought of as a bit-flip operation.

$|q\rangle$ — $\boxed{X}$ —

$$X = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}, \qquad X|0\rangle = |1\rangle, \quad X|1\rangle = |0\rangle.$$

3) Pauli-Y Gate: The Pauli-Y Gates performs a π radian rotation of a qubit state around the Y-axis.

$|q\rangle$ — $\boxed{Y}$ —

$$Y = \begin{pmatrix}0 & -i\\ i & 0\end{pmatrix}, \qquad Y|0\rangle = i|1\rangle, \quad Y|1\rangle = -i|0\rangle.$$

4) Pauli-Z Gate: The Pauli-Z Gates performs a π radian rotation of a qubit state around the Z-axis. Usually this gate can be thought of as a phase-flip operation, since it flips the $|1\rangle$ phase by π radian.

$|q\rangle$ — $\boxed{Z}$ —

$$Z = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}, \qquad Z|0\rangle = |0\rangle, \quad Z|1\rangle = -|1\rangle.$$

5) Rotation Gate: The rotation gates R_x, R_y and R_z respectively performs continuous rotation in the x-axis, y-axis, and z-axis.

$|q\rangle$ — $\boxed{R_x(\theta)}$ —

$$R_x(\theta) = \begin{pmatrix}\cos(\theta/2) & -i\sin(\theta/2)\\ -i\sin(\theta/2) & \cos(\theta/2)\end{pmatrix},$$

$$|q\rangle \text{ —} \boxed{R_y(\theta)} \text{—}$$

$$R_y(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix},$$

$$|q\rangle \text{ —} \boxed{R_z(\theta)} \text{—}$$

$$R_z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}.$$

6) S Gate: The S Gate add a relative phase of $\pi/2$ to the $|1\rangle$ basis of a qubit state.

$$|q\rangle \text{ —} \boxed{S} \text{—}$$

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \qquad S|1\rangle = i|1\rangle.$$

7) T Gate: The T Gate add a relative phase of $\pi/4$ to the $|1\rangle$ basis of a qubit state.

$$|q\rangle \text{ —} \boxed{T} \text{—}$$

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}.$$

8) Controlled-NOT (CNOT / XOR) Gate: The CNOT gate operate on two qubits instead of one, one qubit is designated as the control qubit and the other is the target qubit. It flips the target qubit if the control qubit is $|1\rangle$. Usually the most significant qubit (most left) is the control qubit. The nature of this operation is almost similar to classical XOR operation.

$$|q_0\rangle \text{ —●—}$$
$$|q_1\rangle \text{ —⊕—}$$

$$CNOT = XOR = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

$$CNOT|10\rangle = |11\rangle,$$
$$CNOT|11\rangle = |10\rangle,$$
$$CNOT|00\rangle = |00\rangle.$$

9) Controlled-Z (CZ) Gate: The CZ gate operate with two qubits with a control qubit and target qubit. It will do a sign flip on the target qubit if the control qubit is a $|1\rangle$.

$$|q_0\rangle \text{ —●—}$$
$$|q_1\rangle \text{ —} \boxed{Z} \text{—}$$

$$CZ = CSIGN = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix},$$

$$CZ|10\rangle = -|10\rangle,$$

$$CZ|11\rangle = -|11\rangle,$$

$$CZ|00\rangle = |00\rangle.$$

10) SWAP Gate: The SWAP gate operates on two qubits and it will swap the first qubit with the second qubit. It can be easily seen that this operation is symmetrical.

$$|q_0\rangle \text{ —×—}$$
$$|q_1\rangle \text{ —×—}$$

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$SWAP|10\rangle = |01\rangle.$$

11) Hadamard Gate: The Hadamard gate operate on a single qubit. It can be used to create or collapse a superposition state of a qubit. For example, recall that the $|0\rangle$ state means that a qubit has 100% probability to be measured as 0, the Hadamard gate will turn $|0\rangle$ into $|+\rangle$ which has 50% chance of being measured as 0 and 50% chance of being measured as 1. If we instead apply it to $|+\rangle$ then it will turn into $|0\rangle$, collapsing the superposition.

$$|q\rangle \text{ —} \boxed{H} \text{—}$$

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

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle,$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle,$$

$$H|+\rangle = H(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)) = |0\rangle,$$

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

Recall that the $|-\rangle$ and $|+\rangle$ state has the same measurement probability but differ in relative phase. If Hadamard gate is applied to remove its superposition, even though the measurement probability is the same we will get a different final state result. This signify the importance of relative phase.

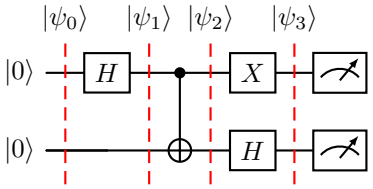
H. Quantum Circuits

Quantum circuit is one way to describe a quantum computer state and operation as a mathematical model. It is a simpler and easier way to describe a quantum computer rather than using the quantum Turing machine model [1].

The concept of quantum circuits is similar to classical Boolean circuits, however there is a difference in how information is represented. In quantum circuits, rather than using standard 0 and 1 (True and False) state, we uses *qubits*.

Quantum circuits consist of a set of quantum register and quantum logic gates arrange in a special way. We usually measure a qubit at the end of the circuit, however we can also have an *ancilla* qubit, which is a qubit for indirect measurement of information without it being measured explicitly. Barriers are often used in quantum circuits to visually and conceptually separate different stages of computation.

Let's go through an example with the following circuit which has a two-qubit register,



Initial state of the register:

$$|\psi_0\rangle = |00\rangle$$

Applying a Hadamard gate to the first qubit gives,

$$|\psi_1\rangle = (H\,|0\rangle)\,|0\rangle = (\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle))\,|0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$$

Then we apply the CNOT gate with first qubit as control qubit,

$$|\psi_2\rangle = CNOT(\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Then we apply the Pauli-X gate to first qubit, and Hadamard gate to the second qubit,

$$X\,|0\rangle = |1\rangle\,, \qquad X\,|1\rangle = |0\rangle\,,$$

$$H\,|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad H\,|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle),$$

Which gives the final state,

$$|\psi_3\rangle = \frac{1}{2}\,(|00\rangle - |01\rangle + |10\rangle + |11\rangle)$$

We then do a measurement operation on the register in which each 2-qubit combination will have a 25% chance of being measured.

I. Quantum Fourier Transform (QFT)

The quantum Fourier transform (QFT) is a quantum circuit, it is a linear operator which acts on a vector state $|q\rangle = |q_0 q_1 \dots q_{2^n-1}\rangle$ of size 2^n and transform it such that the information inside the vector state is encoded in the relative phase of the result in form of the discrete fourier transform (DFT) series. It is defined mathematically as,

$$QFT\;|q\rangle = \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi i\, q_k/2^n}\;|k\rangle\,.$$

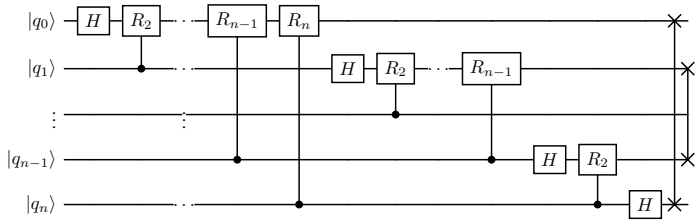
For example we can see the following QFT transformation result in form of tensor product,

$$\begin{aligned} QFT\,|1101\rangle = &\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\,(\frac{|1\rangle}{2})}\,|1\rangle\right) \\ &\otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\,(\frac{|0\rangle}{2}+\frac{|1\rangle}{4})}\,|1\rangle\right) \\ &\otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\,(\frac{|1\rangle}{2}+\frac{|0\rangle}{4}+\frac{|1\rangle}{8})}\,|1\rangle\right) \\ &\otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\,(\frac{|1\rangle}{2}+\frac{|1\rangle}{4}+\frac{|0\rangle}{8}+\frac{|1\rangle}{16})}\,|1\rangle\right) \end{aligned}$$

To implement this circuit, we have to define a new special phase gate named R_k gate which is defined as follows,

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{pmatrix}$$

Then, the general circuit implementation of QFT for state vector of size n can be realised with the controlled-R_k gate, Hadamard gate, and SWAP gate as follows,



The time complexity of applying this circuit is $O(n^2)$. It may be easy to see why, by examining the above circuit. There also exist the inverse quantum Fourier transform (IQFT) circuit which satisfy,

$$IQFT \cdot QFT\,|q\rangle = |q\rangle$$

For any state vector $|q\rangle$. Since all operation in the QFT circuit is reversible, the IQFT circuit can be implemented by mirroring the QFT circuit.

J. Quantum Phase Estimation (QPE)

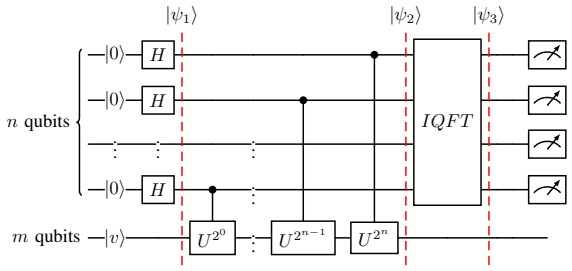
Quantum Phase Estimation (QPE) is an algorithm used to estimate the phase ϕ associated with an eigenvalue of a unitary operator. Recall that a unitary gate operator U is just a matrix, if we take eigenvector v of U with eigenvalue $e^{(2\pi i)\phi}$ then we

can say that $|v\rangle$ is the eigenstate of U. Given a unitary operator U and an eigenstate $|v\rangle$ such that,

$$U|v\rangle = e^{(2\pi i)\phi}|v\rangle$$

QPE can estimates the value of ϕ up to n qubits of accuracy for a choosen n.

The circuit for QPE can be implemented using controlled-U gate, Hadamard gate, and IQFT operation as follows,



We prepare one register with n qubits initalize as the $|0\rangle$ state, and one register with the content of eigenstate $|v\rangle$ which has m qubits. At state $|\psi_1\rangle$, we initiate the superposition state for all qubits in the first register.

Then for each qubit, we will apply the controlled-U to $|v\rangle$ with the qubit as control qubit. Due to a phenomenon known as *phase kickback* which can happen when a unitary operator is applied to its own eigenstate, the applied phase is "kicked back" into the control qubit instead of applied normally to the target qubit. Since the qubit in the first register is used as control qubit, the phase is applied to that qubit (note that U^k means that the operator U is applied k times). The result is that we will get the following,

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{(2\pi i k)\phi}|k\rangle \otimes |v\rangle$$

We know that $|v\rangle$ is the value in the second register, and the rest is the first register. If we factor out the $|v\rangle$, then we can rearrange terms in the value of first register such that it will be exactly identical to the result of a QFT operation. So by applying the $IQFT$ to the first register, we will be able to get the approximation of ϕ up to n qubits accuracy,

$$|\psi_3\rangle \approx |\phi\rangle$$

K. Harrow-Hassidim-Lloyd Algorithm

The Harrow-Hassidim-Lloyd Algorithm is an quantum algorithm which propose a solution to the linear system problem (LSP) in time complexity about $O(logN)$ with some small constant [6].

For a linear system problem,

$$Ax = b$$

This algorithm can be implemented using the Hadamard Gate, $R_y(\theta)$ gate, and controlled-unitary gate which consist of the unitary operator,

$$e^{iAt}$$

Where t is a specific relative phase value choosen by its encoding scheme [7]. The encoding scheme depends on the ratio of the eigenvalue of A. For Example, if the ratio of such eigenvalue is $2:1$, then we would say that the encoding scheme would be $|10\rangle$ and $|01\rangle$.

The output of this algorithm is not the vector x itself, but rather the ratio of the square of each element of x. For example for 2 by 2 matrix A, the output of this algorithm will be $|x_0|^2 : |x_1|^2$ where the ratio is encoded inside the qubits which was determined when we get the encoding scheme.

This algorithm require 3 register, which is one ancilla register, one n qubits register where n is as big as possible, and one n_b qubits register where n_b is the length of the vector b.

III. Implementation

A. Constraint

Due to the many constraint of the HHL quantum algorithm, the author will limit this implementation of testing for the linear system problem $Ax = b$ for A of size 2 by 2 and b of size 2 by 1. Some research has concluded that higher size of A and b might require signifcant effort [6] [7] [8].

B. Programming Language and Supporting Tools

We will use the Qiskit quantum circuit simulator and Qiskit-Aer quantum computer simulator to design and simulate the HHL quantum algorithm. We will also use the numpy array utility to interact with number with more precision, time library to measure duration, and we will use the scipy.linalg builtin linear algebra library to do our Gauss-Jordan elimination.

```
import time
import numpy as np
from scipy.linalg import expm
from qiskit import QuantumCircuit,
↪  QuantumRegister, ClassicalRegister,
↪  transpile
from qiskit.circuit.library import RYGate,
↪  UnitaryGate
from qiskit_aer import AerSimulator
```

C. Implementation Detail

For testing, we will generate manually 10 test cases of the linear system $Ax = b$ with custom constraint, then for each test we will measure the time taken between the operation of Gauss-Jordan (classical algorithm) dan the HHL (quantum algoritm). After that, for the linear system problem result x, we will measure $|x_0|^2 : |x_1|^2$ since that is the default output of the HHL algorithm. We will then treat the Gauss-Jordan result as our control sample and measure the error of our HHL implementation.

Our technical implementation detail will be heavily inspired by [7] [8] and we will merely extend the implemention in the referenced paper such that we can test arbitrary linear

system problem according to our constraint. Here is the HHL algorithm implementation,

```python
def HHL(A, b):

    # Defining the registers
    nc = 2
    clock = 2
    ancilla = 1
    qr_clock1 = QuantumRegister(1, 'clock_1')
    qr_clock2 = QuantumRegister(1, 'clock_2')
    qr_b = QuantumRegister(1, 'b')
    qr_ancilla = QuantumRegister(ancilla,
    ↪  'ancilla')
    cl = ClassicalRegister(nc, 'cl')
    qc = QuantumCircuit(qr_ancilla, qr_clock1,
    ↪  qr_clock2, qr_b, cl)

    # Finding appropriate encoding scheme
    t, qubits1, qubits2 =
    ↪  find_encoding_scheme(A, nc)
    assert t, "No valid encoding scheme
    ↪  found!"

    # Constructing the controlled unitary gate
    U1 = expm(1j * A * t)
    U2 = expm(1j * A * 2 * t)
    gate_u1 = UnitaryGate(U1,
    ↪  label="exp(iAt)")
    gate_u2 = UnitaryGate(U2,
    ↪  label="exp(iA2t)")

    # Applying state preparation
    qc.x(qr_b)
    qc.barrier()

    # Applying the uniformly distributed
    ↪  superposition
    qc.h(qr_clock1)
    qc.h(qr_clock2)
    qc.barrier()

    # Applying the controlled unitary
    qc.append(gate_u1.control(1), [qr_clock1,
    ↪  qr_b[0]])
    qc.append(gate_u2.control(1), [qr_clock2,
    ↪  qr_b[0]])
    qc.barrier()

    # Applying IQFT
    qc.h(qr_clock2)
    qc.cp(-np.pi/2, qr_clock1, qr_clock2)
    qc.h(qr_clock1)
    qc.swap(qr_clock1, qr_clock2)
    qc.barrier()

    # Applying the controlled rotation on the
    ↪  ancilla
    qc.cry(np.pi, qr_clock1, qr_ancilla[0])
    qc.cry(np.pi/3, qr_clock2, qr_ancilla[0])
    qc.barrier()

    # Applying the QFT
    qc.swap(qr_clock1, qr_clock2)
    qc.h(qr_clock1)
    qc.cp(np.pi/2, qr_clock1, qr_clock2)
    qc.h(qr_clock2)
    qc.barrier()
    qc.measure(qr_ancilla, cl[ancilla:])
    qc.barrier()

    # Applying the inverse controlled unitary
    qc.append(gate_u2.inverse().control(1),
    ↪  [qr_clock2, qr_b[0]])
    qc.append(gate_u1.inverse().control(1),
    ↪  [qr_clock1, qr_b[0]])
    qc.barrier()

    # Collapse superposition, measure and
    ↪  return circuit with its encoding
    ↪  scheme
    qc.h(qr_clock1)
    qc.h(qr_clock2)
    qc.barrier()
    qc.measure(qr_b, cl[:ancilla])
    return qc, qubits1, qubits2
```

Notice that this implementation heavily mirrors the referenced paper, however this impelementation extend the referenced impelementation to have proper controlled-unitary gate, rather than hardcoded one. And we impelement custom linear search to find the correct encoding scheme before the algorithm begin.

```python
def find_encoding_scheme(A, n):

    # Get the eigenvalue
    evals = np.linalg.eigvalsh(A)
    N = 2**n
    MAX_ITER = 25000

    # Iterate as many times as we can to find
    ↪  the correct 't' such that an integer
    ↪  encoding scheme is found, and unique
    ↪  between 1..2**(n-1)
    for i in range(1, MAX_ITER):
        t = i * np.pi / 4
        scaled = (N * evals * t) / (2 * np.pi)
        rounded = np.round(scaled).astype(int)

        if np.allclose(scaled, rounded) and
        ↪  np.all((rounded > 0) & (rounded <
        ↪  N)) and len(set(rounded)) ==
        ↪  len(rounded):
            encoding = [format(v, f'0{n}b')
            ↪  for v in rounded]
            return t, encoding[0], encoding[1]

    return None, None, None
```

With all the function implemented, we can define a driver code as folows:

```python
if __name__ == "__main__":
    for idx, test in enumerate(test_cases):

        A_input, b_input = test["A"],
        ↪  test["b"]
        print(f"Test case {idx + 1}:\nA =
        ↪  {A_input}\nb = {b_input}\n",
        ↪  end='')

        # Classical algorithm
        t1 = time.perf_counter()
        gauss_jordan_result =
        ↪  np.linalg.solve(A_input, b_input)
        t2 = time.perf_counter()
        actual_ratio = gauss_jordan_result[0]
        ↪  ** 2 / gauss_jordan_result[1] ** 2
```

```
        print(f"Actual Ratio (Gauss-Jordan Elimination): {actual_ratio} ({t2 - t1} ms)")

    try:
        t1 = time.perf_counter_ns()

        # Construct the quantum circuit and encoding scheme
        hhl_qc, qubits1, qubits2 = HHL(A_input, b_input)

        # Run in Aer-simulator
        simulator = AerSimulator()
        new_circuit = transpile(hhl_qc, simulator)
        job = simulator.run(new_circuit, shots = 1000000)
        result = job.result()
        counts = result.get_counts()
        total_shots = sum(counts.values())
        probabilities = {state: count / total_shots for state, count in counts.items()}
        plot_distribution = (counts)

        # Get the approximate ratio
        prob_x1 = counts[qubits1]
        prob_x2 = counts[qubits2]
        approx_ratio = prob_x1**2 / prob_x2**2

        t2 = time.perf_counter_ns()
        print(f"Approximated Ratio (HHL Algorithm): {approx_ratio} ({t2 - t1} ms)\n\n")

    except AssertionError:
        print("Approximated Ratio (HHL Algorithm): Unable to find valid encoding scheme for this test case!\n\n")
```

IV. Result and Discussion

We can see that our classical algorithm beats the proposed quantum algorithm by time, The huge error which occur between the classical result and the quantum result is most likely due to the faulty way of finding the encoding scheme, since it is improvised greatly by the author.

Recall that the Gauss-Jordan algorithm has time complexity $O(N^3)$ and the HHL algorithm has time complexity of about $O(log(N))$ with some coefficient [6]. This seems contradictionary with our implementation, since we run this on a quantum computer simulation and therefore there will be virtual machine overhead. Also there is likely huge amount of inefficient overhead due to author way of impelementation.

V. Conclusion

While the world of quantum computing is promising, especially with algorithm like the Harrow-Hassidim-Lloyd algorithm. Implementation of such quantum algorithm still unable to beat the classical equivalance, even on noise-less ans error-less simulator. The many details of theory also can lead to imperfect and inefficient implementation of the quantum algorithm, therefore it is still not ready for a general use algorithm.

Test	A	b	Actual Ratio	HHL Ratio
1	$\begin{pmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{pmatrix}$	$(1, 0)$	9.0	8.8561
2	$\begin{pmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{pmatrix}$	$(0, 1)$	0.1111	9.1540
3	$\begin{pmatrix} 1 & -0.333 \\ -0.333 & 1 \end{pmatrix}$	$(0, 1)$	0.1111	8.9460
4	$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$	$(1, 1)$	1.0	0.00512
5	$\begin{pmatrix} 2.5 & 0.5 \\ 0.5 & 2.5 \end{pmatrix}$	$(2, 1)$	9.0	0.00508
6	$\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$	$(0, 1)$	0.1111	8.9959
7	$\begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$	$(1, 2)$	0.0	0.00512
8	$\begin{pmatrix} 1 & -0.333 \\ -0.333 & 1 \end{pmatrix}$	$(1, 2)$	0.5102	8.9913
9	$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$	$(3, 3)$	1.0	0.00512
10	$\begin{pmatrix} 1.25 & 0.25 \\ 0.25 & 1.25 \end{pmatrix}$	$(0, 2)$	0.04	0.00519

TABLE I
Comparison of classical Gauss–Jordan ratios and HHL observable-based ratios for 2×2 Matrix.

Test	Classical Time (ms)	HHL Simulation Time (ms)
1	7.6×10^{-5}	1.10×10^{9}
2	6.4×10^{-5}	7.50×10^{8}
3	5.7×10^{-5}	6.68×10^{8}
4	6.1×10^{-5}	7.21×10^{8}
5	4.1×10^{-5}	7.41×10^{8}
6	4.0×10^{-5}	7.05×10^{8}
7	4.1×10^{-5}	7.08×10^{8}
8	4.0×10^{-5}	7.23×10^{8}
9	5.2×10^{-5}	7.11×10^{8}
10	4.3×10^{-5}	7.47×10^{8}

TABLE II
Runtime comparison between classical Gauss–Jordan elimination and HHL quantum circuit simulation.

VI. Appendix

Source Code of Experimentation (Github): https://github.com/FieryBanana101/HLL-Analysis

Acknowledgment

The author is extremely grateful to God Almighty for providing strength, determination, and opportunity to bring this paper to completion. The author also expresses very deep respect and appreciation to **Dr. Ir. Rinaldi Munir, M.T.** and **Ir. Rila Mandala, M.Eng., Ph.D.**, the lecturer of the IF2123 Linear Algebra and Geometry course, for their dedicated guidance and support, which have been a source of inspiration throughout his teaching journey with the students. The author also acknowledge the use of Generative AI in the making of this paper, not to help develop the core analytic of the paper, but to help find initial inspiration, overall structure of literature

review and to help find language syntax, theorem definition, or internet resource.

REFERENCES

[1]
[2]
[3] Artur et al. "Basic concepts in quantum computation" 2024. [Online]. Available: https://arxiv.org/pdf/quant-ph/0011013 [Accessed: Dec. 22, 2025]
[4] Catt, Elion, Markus Hutter. "A Gentle Introduction to Quantum Computing Algorithms with Applications to Universal Prediction" 2020. [Online]. Available: https://arxiv.org/pdf/2005.03137 [Accessed: Dec. 22, 2025]
[5] Morales et al. "Quantum Linear System Solvers: A Survey of Algorithms and Applications" 2025. [Online]. Available: https://arxiv.org/pdf/2411.02522 [Accessed: Dec. 22, 2025]
[6] Harrow et al. "Quantum algorithm for linear systems of equations" 2009. [Online]. Available: https://arxiv.org/pdf/0811.3171 [Accessed: Dec. 22, 2025]
[7] Zaman et al. "A Step-by-Step HHL Algorithm Walkthrough to Enhance Understanding of Critical Quantum Computing Concepts" 2023. [Online]. Available: https://arxiv.org/pdf/2108.09004 [Accessed: Dec. 22, 2025]
[8] Galvao et al. "Solving Linear Systems of Equations with the Quantum HHL Algorithm: A Tutorial on the Physical and Mathematical Foundations for Undergraduate Students" 2025. [Online]. Available: https://arxiv.org/pdf/2509.16640 [Accessed: Dec. 22, 2025]

STATEMENT OF ORIGINALITY

I hereby declare that the paper I wrote is my own writing, not an adaptation or translation from other people's papers, and not plagiarism.

December 24 2025,

Name/NIM:
Muhammad Fatih Irkham Mauludi / 13524004