An Implementation of a Quantum Algorithm for Solving Linear Equations

S.A. Sigurdsson

An Implementation of a Quantum Algorithm for Solving Linear Equations

by

S.A. Sigurdsson

Interim Thesis
7. March 2020

Student number:	5162599	
Project duration:	September 1, 2019 – March 1, 2020	
Thesis committee:	Prof. M. Moller,	TU Delft, supervisor
	Prof. C. Vuik,	TU Delft

Preface

I am not a Physicist nor am I a Computer Scientist, I studied Mechanical Engineering at Reykjavik University and worked in mechanical design before starting this Masters program that led to this thesis. Therefore my interest in the topic of Quantum Computers comes not from an interest in theorizing better algorithms, but from an interest in getting better ways to solve practical problems. As I didn't have a background to fully understand the papers I needed as the foundation for the practical implementation of this thesis I read, and I read a lot. Assuming you the reader comes from a similar position as me when I started I will try to cover well and explain as best I can those topics that I thought most important to this work. However I can not cover everything and at times I will be vague, this will especially apply to the topic of quantum physics and any practical implementation of Quantum Computers. So for a good introduction to these topics, I would like to recommend the lecture series from Richard Feynman, published online by the California Institute of Technology [9], as well as the book Quantum Computing: a gentle introduction by Eleanor Rieffel and Wolfgang Polak [6].

S.A. Sigurdsson
Delft, March 2020

Contents

1 Introduction

In 1883 Antoni Gaudi was designing his masterpiece La Sagrada Familia he had a problem, how to calculate the intricate archways and columns that he had envisioned for his design. We today can set this up in a computer and analyze it with e.g. the Finite Element method, then compute the model and get an answer. These methods could not have worked for Gaudi as he was doing this well before the Galerkin method was set forth and even further from a computer that could have done the calculations necessary.

So how did Gaudi solve this? As he was using an old building style where the entire structure is kept under compression, so that there is no need for steel for structural support, and only his chosen material of Catalonian sandstone would be used. He devised a way of using ropes hanging from the ceiling and connected them in such a way that they formed the building he wanted, he then hung weights on the ropes where they would be supporting the roof of the building. As he hung the ropes and weights the calculations for the archways and columns were instantly done as the rope itself solves the ideal arch, as shown by Robert Hook, as it always hangs in perfect tension and is under the same force was acting on the rope as would be acting on the building, gravity. This a fundamental law the rope follows and the gravity the building would be standing under was all he needed to complete the calculations, now Gaudi could measure the archways on his model and scale them up to the real size of the model, a replica of his model stands in the Sagrada Familia today, see figure 1.1, [1].

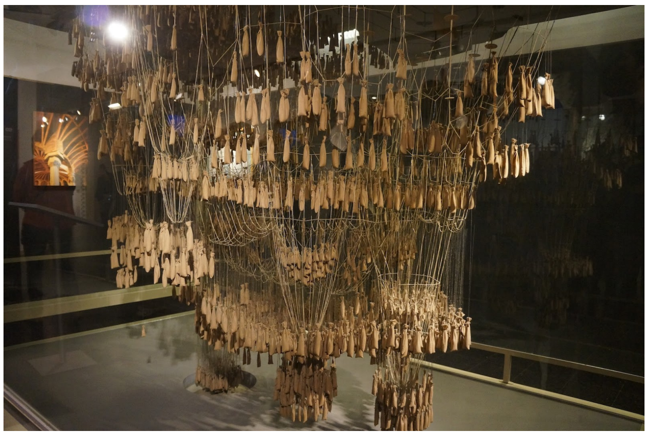
Figure 1.1: A replica of Gaudis model from La Sagrada Familia [12]

This story is to illustrate the way of Analog computers and to try to set the frame of mind that a complex computation does not need the digital machines we are so used to today. In a more abstract notion of computation, the information in the numbers is like currency, that is they are the medium we use to exchange information between systems like we use our currency to exchange for goods and services. In Guidi's system a hanging rope was used to represent an archway and a sandbag the weight it would need to support, then gravity did the calculations for him. With measurements of the model he then turned it into numbers and scaled them properly to represent the real version. Using this exchange view we see that something is fundamentally the same about the model and how we move it into the real world, this we can call *information*.

1.1. Information

Information theory came about in the late 1940s and was first introduced by Claude Shannon [15], mainly in relation to communication and computation.It has now grown from its origin into a fundamental part of physics as we can say that every microscopic state a physical system can be characterized by ho much information is needed to define that microscopic state and it was perhaps this definition that led the physicist John Wheeler to say ”Information is the most fundamental building block of reality [11].”

Shannon’s information theory focused on information as an abstraction and a distinction between possible alternative messages, not the meaning of the message. Now this means that a message is one out of a number of alternative possible messages and to be able to reliably distinguish between the alternatives is the information. From this focus on the distinction of possible messages, Shannon theorized the smallest possible level of distinction of information was a binary one, the 0 and 1, and he called this the **bit**. To then capture the information in a message we can try to think of it as how many yes or no questions you would need to ask to determine the message, this is also known as the message entropy. From the bit, we can build and send any message we want by stringing together enough bits and we can transform the information and do calculations.

Our classical computers user the bit as the bases of their calculations and the representation of information, this is done by a string of bits with a given size represents a number or a letter and calculation are done on the bits them selves in the computer using logic gates. So the informational value of a 8-bit number is obvious, it is simply 8 bits and so that is all the information you can work with at any given moment, this is not true for quantum information. As we will explore further in later chapters quantum information is represented by the qubit and it can represent more than one bit of information at a time using the natural effects of quantum mechanics, as a comparison a 8-qubit string will have 2^8 bits of information.

1.2. Universal Computers

Alan Turing, one of the fathers of modern computing, reasoned that all sufficiently sophisticated computers are equivalent. That is once a computer is able to simulate another computer all computations done on the second can be done on the first, it might not be as efficient but in a conceptual way you now try to capture the concept of a universal computer, that is a computer that is equivalent to all others. Turing set to design the simplest such a computer possible and this came to be known as the Turing machine.

His model consisted of a machine that had infinite tape to read and write to, a head that executed the reading and the writing, the machine has finitely many states (memory), and a transition function δ. The transition function determines what the machine will do at time t, given the current state it is in and what is read at time t; The δ functions output is what will be written on the tape, what direction the head will move, and what will be the state of the machine at time $t+1$. To see that such a simple machine is a universal computer that can, in fact, simulate any other computer is difficult,

The theorem says that all sufficiently powerful computers are the same, though this only means in that they can do the same calculations the important factor when you have a universal computer is time, how long does it take to finish the calculation. This is called the complexity of an algorithm and we categorize problems according to how the computation time scales with the input size of the problem. In computers, this is also analyzed as the number of operations a computer must perform to do the calculation. Simple problems such as just addition and multiplication are polynomial in scale, called P class, that is we know an algorithm that can solve these problems on a time scale that is a polynomial of the input. A more complex problem then is more time consuming and so we call them non-deterministic polynomial time problems, NP class, and so for these are problems we only have algorithms that can get to a solution in non-polynomial time. For researchers, the latter class is the most interesting as even though a problem is classed as NP now it doesn’t prove that there can’t be a polynomial algorithm just that we don’t have one yet, this class is where quantum computers come in.

Quantum computers have been shown to be able to crack some of these NP class problems within a polynomial time and so have broken a barrier thought by many to be impregnable, they do so by utilizing quantum information and by forming a universal computer out of quantum systems. Though to be clear a quantum computer, can not calculate anything more than classical computers can. That is any Turing machine can calculate *any* problem and they are just Turing machines like any classical computer. Their benefit is in re-framing certain problems so that they can be solved faster than they could be otherwise. Like Gauids ropes they are just better suited to some types of problems than others. Before we go into more into how quantum computers we need to familiarize ourselves with the notations and conventions of quantum mechanics.

1.3. Quantum Mechanics

Quantum mechanics is a theory of physics that was set out in the early 20th century to explain the behavior of matter on the smallest scales. This was primarily driven by the strange way objects on this scale seem to behave, this is for example the wave particle duality of light. Out of these developments came the essential mathematical framework used to describe this world, these descriptions of quantum systems that we will explore have been thoroughly tested and confirmed in almost 100 years since their inception.

Linear Algebra is the mathematical language used to describe quantum systems. To describe each state we use a two dimensional vector of complex numbers in a Hilbert space, $|\theta\rangle \in \mathbb{C}^2$, the notation $|\theta\rangle$ is called a ket of θ where

$$|\theta\rangle = \begin{bmatrix}\alpha \\ \beta\end{bmatrix}, \quad \langle\theta| = \begin{bmatrix}\alpha^* & \beta^*\end{bmatrix}.$$

This notation is commonly used in quantum physics and is the one we will use for the remainder of this work, it is a compact notation that makes it easy to write common operations like an inner product $\langle\phi|\psi\rangle$. With regular vector multiplication and inner products, we use the operation of tensor products on quantum systems. This is when we connect two qubits in an operation, more on this in the next chapter, an example of a tensor product on two qubit vectors would be

$$\begin{bmatrix}a \\ b\end{bmatrix} \otimes \begin{bmatrix}x \\ y\end{bmatrix} = \begin{bmatrix}ax \\ ay \\ bx \\ by\end{bmatrix}.$$

As a short hand notation we often write $|\phi\rangle\,|\psi\rangle$ as the tensor product of $|\phi\rangle$ and $|\psi\rangle$. Further common concepts from linear algebra are used such as unitary matrices, linear independence, and bases for vector spaces, though these use a common notation and so should be familiar to anyone who has studied those in any other context. These concepts will not be explored in depth here but we will come back to them when we introduce quantum algorithms and how they function.

1.4. Quantum Algorithms

In a quantum system information is stored in a fundamental property of the particle, this can be e.g. the spin of an electron. Though the spin can only be measured in two values, spin up or spin down, the information differs from the classical bit. This is because the electron can be in any state in between these values before it is measured and that is key information that we can use for calculations.

In a sentence, a Quantum computer is simply a computational device based around a quantum process. The benefits of which are inherent in these quantum processes, which can store more information than your typical computer and thus solve more complicated problems with fewer processes. This allows quantum computers to solve some problems faster than our classical computers.

To use this ”extra” information we need special algorithms, we call these Quantum algorithms. They are more analogous to classical algorithms as they are a series of steps that compute a solution to a given setup.

As a classical computer uses bits a Quantum Computer stores it’s data and programs in a qubit and these qubits can be put in a superposition, where n qubits have information on both a 1 state and a 0 state of n values giving a total information capacity of 2^n bits. A quantum algorithm is an algorithm that preserves these quantum states and can use them to simultaneously do the calculation on all of its possibilities, giving it the ability to e.g. run a program with both a 0 and a 1 as input and evaluate both options.

As was talked about in the mathematics of quantum chapter we can describe qubits using two dimensional vectors. Further we can write those vectors as a linear combination of some basis vectors $\alpha_0 |0\rangle + \alpha_1 |1\rangle$ here we are using $|0\rangle, |1\rangle$ as our base, where

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

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

This linear combination is called a superposition of the bases and needs to fulfill the condition that $\alpha_i \in \mathbb{C}$ and $|\alpha_0|^2 + |\alpha_1|^2 = 1$. To draw this complex vector we us a three dimensional shell of a sphere, where the third dimension comes from the complex numbers, in figure 1.2 we see a demonstration of this using a state $|\psi\rangle$.

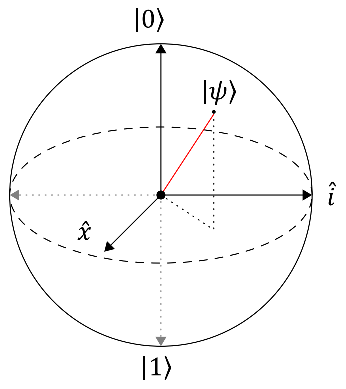

Figure 1.2: Bloch spheare

So while the qubit remains unobservable it is any state on the surface of the Bloch sphere but as it is measured it collapses to either $|0\rangle$ or $|1\rangle$, or to another measurement basis. Though this state is a real physical state of a quantum system it is hard to imagine, so often one only looks at the mathematics, that have been verified extensively for the last century. This is because the mechanics in the quantum realm can be unintuitive and clash with our regular experience of reality. Let's look at an example of a qubit in a state where it has a half chance of being measured in the $|0\rangle$ or $|1\rangle$, $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, this draw on the Bloch sphere is shown in figure 1.3.

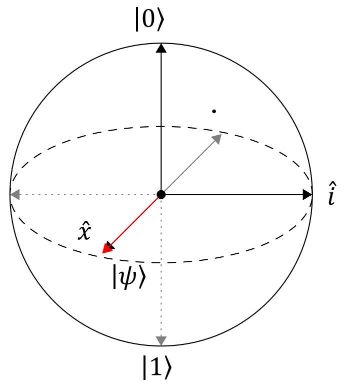

Figure 1.3: Bloch with a half state vector ψ

As you can see this state falls directly on the positive part of the $\hat{x}$ axis, this is special enough case that it is often simply referred to as $|+\rangle$. As you can imagine the cases where the quantum state falls directly on the other axes are also special cases, $|-\rangle, |i\rangle, |-i\rangle$. Note that there is no *measurable* difference between $|+\rangle$ and $|-\rangle$ however, they do not behave the same in computation and this is crucial [6].

1.5. Demonstrating Algorithmic advantage

Now we have talked about how and why quantum computing works and we have gone over the tools and notations we need to understand a quantum algorithm. So when is a quantum algorithm faster? The first example is the Deutsch problem, which was later expanded to the Deutsch and Jozsa's Algorithm. The problem the Deutsch and Jozsa algorithm solves is defined such;

$$x \in \mathbf{Z}_{[0,2^n-1]},$$

$$f(x) : \mathbf{Z}_{[0,2^n-1]} \mapsto \{0,1\},$$

$$\text{and f is either balanced or constant.}$$

Balanced here means that f gives an outcome of 0 for a random halve of the values of x and 1 for the others, constant means that for any value of x it is either 0 or 1. Now a classical algorithm can solve this problem in $\frac{2^n}{2} + 1$ steps to have guaranteed success, that is check half of all possible x values and then one more to be sure.
The quantum algorithm proposed to solve this, by David Deutsch and Richard Jozsa in 1992 [8], can do this in one step.

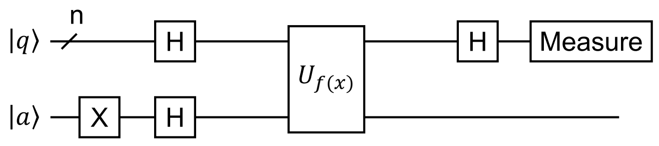

Figure 1.4: Deutsch-Jozsa circuit

To prove this method is not simple but lets try to go through the algorithm step by step. To start we assume all registers are in the $|0\rangle$ position, we need though that the *a* register be in the $|1\rangle$ position so we flip it with a not gate, also called Pauli X. Next we switch our basis from z to x by using the Hadamard gate on all registers. Then our states are

$$|q\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

$$|a\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

In this state we apply is the unitary operator that represents the oracle, we implement it separately by connecting a cnot gate where appropriate from the values in the *q* register and to the *a* qubit. This formulation allows for the final state of the *q* to tell us if the function encoding the unitary is balanced or constant, this is because with a control in the Hadamard basis and with a target bit in $|1\rangle$ phase kickback gives us the state

$$\frac{1}{2^n} \sum_{y=0}^{2^n-1} \left[\sum_{x=0}^{2^n-1} (-1)^{f(x)} (-1)^{xy} \right] |y\rangle .$$

Now we convert the basis back by using the inverse of the Hadamard which is just the Hadamard itself, and measure the $|q\rangle$ register, if it comes out 1 the function is constant and else balanced. So this result is achieved in a single step when a classical method needs $n/2 + 1$, though it should be pointed out that for this specific problem the advantage is by design; It was the specific intent of Deutsch to find a problem that would have an advantage when formulated as a quantum system. This advantage, going from $O(n)$ to $O(1)$, is not restricted to such creations and it is what researchers look for when developing new quantum algorithms. This potential to take a problem down on the complexity scale has been shown in a multitude of other practical problems, from factoring numbers to the optimization. The algorithm that is of interest in this paper is a quantum solution to the problem of solving linear systems. This algorithm has been shown to have better efficiency than any classical one and so it has great potential that we will explore in the next chapter, 2.

2 Quantum Linear Solver

A linear system of equations is a common problem known to all branches of science, it is frequently a part of, design problems, signal processing, and homework. It is the way we solve a multitude of connected variables in a single system and the problem is formulated such

Linear system of equations
Given an invertible matrix $A \in \mathbb{C}^{N,N}$, a vector $b \in \mathbb{C}^N$ find a vector x given

$$Ax = b. \tag{2.1}$$

This problem is solved by finding, or approximating, an inverse of the matrix A such that $x = A^{-1}b$. Many algorithms exist to solve this problem the one generally considered most efficient is the method of Conjugate Gradients it can solve this in $O(N\sqrt{(}\kappa))$ time, CG has the condition that A be a Hermitian, positive-definite matrix.

For solving the linear system in a quantum setting a change is needed to reach the full benefits of quantum speedup. The general formulation is:

Quantum Linear Solver
Let A be an $N \times N$ Hermitian matrix with a unit determinant. Let b and x be N-dimentional vectors such that $x := A^{-1}b$. Let the quantum state on $\lceil log(N) \rceil$ qubits |b> be given by

$$\frac{\sum_i b_i |i>}{|| \sum_i b_i |i> ||_2}$$

and for |x> by

$$\frac{\sum_i x_i |i>}{|| \sum_i x_i |i> ||_2}$$

where b_i, x_i are the i-th components of b and x respectively. Given A and |b>, output a state $|\tilde{x}>$ such that $||(|\tilde{x}> - |x>)||_2 \leq \epsilon$, with some probability larger than 1/2.

$$|x\rangle = A^{-1}\,|b\rangle \tag{2.2}$$

The output $|x\rangle$ is a quantum state that represents x and to read out all N values of x would take $O(N)$ runs of the algorithm. This is because even though $|x\rangle$ contains all the information of x we can measure only one value of it each time, so to use this solution we must remain in the quantum state to be useful. To use this then we don't simply look at a $Ax = b$ problem but embed this as a routine in another problem would where e.g. we calculate $x^T Mx$, where M is some linear operator, you can map M to a quantum operator and get $\langle x| M\,|x\rangle$, such a problem is further discussed in 3.2.

2.1. Quantum algorithm for linear systems of equations

In 2009 Harrow, Hassim and Lloyd, [HHL [10]], introduced a Quantum Algorithm to solve the quantum variation of the linear system $Ax = b$. Their solution is built from previously established quantum algorithms, the three main steps of the HHL algorithm are the phase estimation, ancilla rotation and then uncomputing of the registers B and C, this is highlighted in the circuit model of the algorithm in figure 2.1.

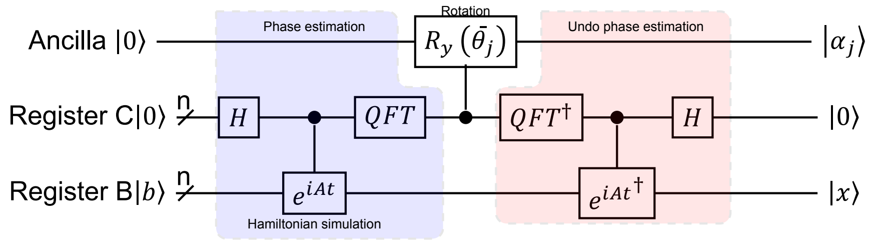

Figure 2.1: Circuit model of the algorithm proposed by Harrow, Hassim and Lloyd [13]

In their paper Harrow, Hasim, and Lloyd use a more detailed eight step procedure to describe their algorithm as they include the initialization steps needed to get b and A into the correct states,

1. Converting A into a Unitary operator e^{iAt}.
2. Prepare the b register $|b\rangle$.
3. Compose $|b\rangle$ into the eigenvector basis of e^{iAt} using phase estimation.
4. Apply the conditional Hamiltonian evolution, conditioned on register C in 2.1.
5. Apply the Quantum Fourier Transform on register C.
6. Use an additional ancilla, α_j, with a rotation conditioned on C register.
7. Undo the phase estimation to uncompute everything except the last ancilla bit.
8. Measure α_j, if it is 1 the computation was successful if not repeat.

Phase estimation

This part estimates the eigenvalues of A, by taking operator eigenstate of A and estimates the eigenvalues. This is broken into two major subroutines the Hamiltonian Simulation and the Quantum Fourier Transform.

Hamiltonian Simulation

Converting A into a unitary operator e^{iAt} is conditioned on A being Hermitian, this has been shown to be efficient through multiple different techniques. All these techniques are a form of Hamiltonian simulations, this is defined as a Hamiltonian that acts on n qubits can be efficiently simulated by a U_H quantum circuit if for a $O(poly(n,t,1/\epsilon))$ number of gates $||U_H - e^{iHt}|| < \epsilon$ holds. Here we have added a dependency on t which is important as it can be shown that any such simulation requires $O(t)$ time to calculate. An example of a method that does this is the Trotter-Suzuki method, details of this method can be found in [1], this method has complexity

QFT

The Fourier transform is a method that allows you to change to frequency space and analyze a signal on its base frequencies. Given a square invertible matrix F it's discrete Fourier transform is $\mathcal{F}_{nm} = \frac{1}{\sqrt{N}} e^{(\frac{i2\pi}{N})(nm)}$, as the columns of the new matrix are orthonormal they can be used as a basis known as the Fourier basis of the matrix F. The same way as the classical Fourier transform works the quantum version, QFT, takes a quantum state $|x\rangle$ to a new state f_x thus,

$$|x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{N-1}^{k=0} e^{\frac{i2\pi}{N}xk} |k\rangle$$

Rotation

The purpose of the ancilla qubit is to check if the estimation of the eigenvalues was successful in the phase estimation so in the end we only need to measure the ancilla qubit and we can know if we should run everything again or if we should continue with $|x\rangle$ into the next step. This check is done using amplitude amplification which is an extension of Grover's search algorithm.

Uncomputing

As gates in quantum circuits are unitary operation to invert the operations already done we simply take the conjugate transpose and we will have a valid gate to invert it back.

Completion

Measuring the ancilla qubit is the only measurement necessary for the algorithm and it determines whether the algorithm worked or not, however as the condition is before the undoing the phase estimation an imperfect step there would not be detected by this method.

2.2. Variations and improvements

Improvements have been made both in the efficiency of the algorithm and in relaxing restrictions, which is improving it for dense matrices and dropping the number of qubits needed for realization.

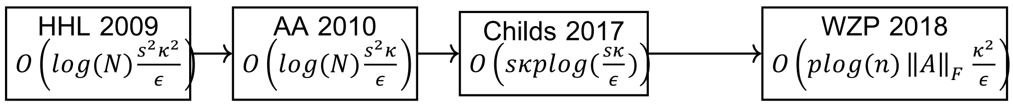
HHL 2009
$O\left(log(N)\frac{s^2\kappa^2}{\epsilon}\right)$
AA 2010
$O\left(log(N)\frac{s^2\kappa}{\epsilon}\right)$
Childs 2017
$O\left(s\kappa plog(\frac{s\kappa}{\epsilon})\right)$
WZP 2018
$O\left(plog(n)\,\|A\|_F\,\frac{\kappa^2}{\epsilon}\right)$

Figure 2.2: Improvements on HHL through the years

As the original implementation is the focus of this paper I will not go into details on these improvements but briefly go over them.

Faster algorithm for solving systems of linear equations*2010*
Andris Ambainis was the first to improve on the work of Harrow, Hassim, and Lloyd by improving the runtime from $O(\kappa^2 log(N))$ to $O(\kappa log^3(\kappa)log(N))$.

Quantum circuits for solving linear systems of equations *2011*
In 2011 Cao, Daskin, Fankel, and Kais at Purdue University, proposed an improved circuit to the HHL model [5]. This model runs in $O(log(N)\frac{\kappa^2}{\epsilon})$ time, where κ is the condition number $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$ and ϵ is the error in the output of the x register.

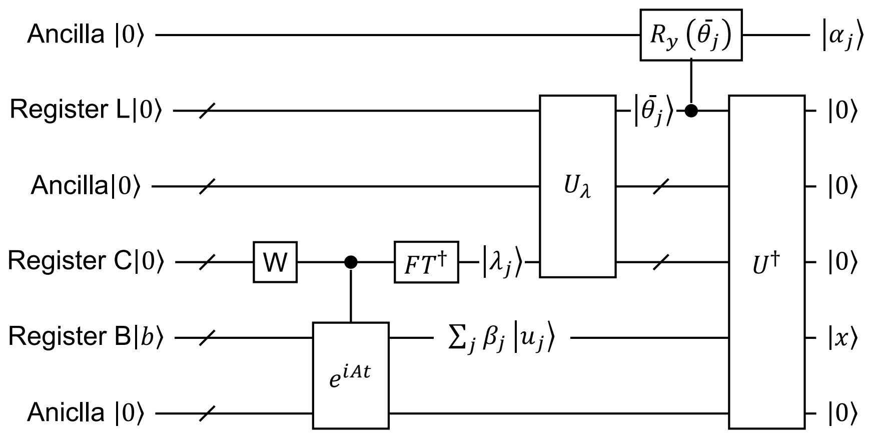

Figure 2.3: From [13]

Experimental realization of HHL
In 2011 Pan, Cao, et al. ran a proof of concept experiment usin a 4-qubit nuclear magnetic resonance quantum information processor to solve a 2×2 linear system using the HHL algorithm [13]. Their experiment was successful and results showed a 96% fidelity, this was the first realization of the HHL algorithm.

Similarly in an article from 2013 by Cai, Weedbrook, et al. the simplest meaningful instance of the HHL algorithm was tested, 2×2 matrices, was tested again in an experiment using a linear optical network with four photon base qubits [4].

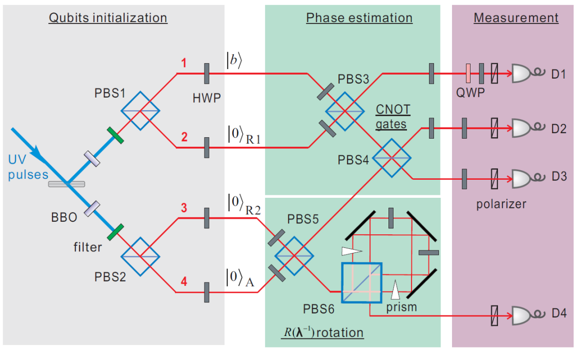

Figure 2.4: The experimental circuit used four qubits to solve a 2x2 system[4]

Variational Quantum Linear Solver 2019

VQLS is a hybrid solution using both classical and quantum computing methods to solve the system of equations [3]. This new algorithm, published 2019, is not an iteration on the HHL but a proposed intermediary solution to HHL high demand of qubits and high quality of computation. This method is designed to work on so called noisy intermediate scale quantum computers (NISQ) by reducing the depth of the quantum circuit needed to solve the problem. It does this essentially by moving parts of the algorithm back to a classical computer, this frees up qubits and creates stability as they do not need to be in a quantum state for too long, reducing the likelihood of de-coherence. The authors of the article immediately experimented with this new algorithm on hardware from Rigetti using a problem size of 32×32 with successful results. This change in the algorithm comes at a cost though as this algorithm is the only one we have looked at here that has worse scaling behavior than HHL though it still remains efficient in κ and $1/\epsilon$.

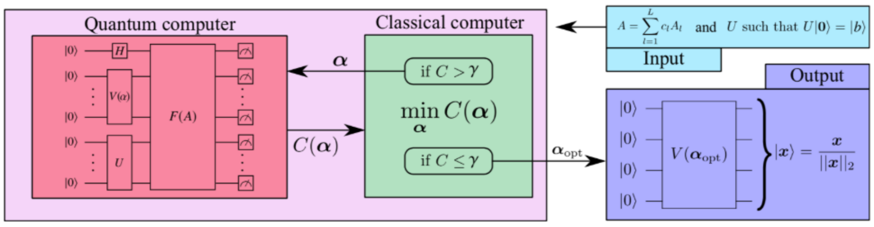

Figure 2.5: A circuit representation of the VQLS method, from Bravo et al [3]

3 Implementation

Implementations of a QLSA have come far since Harrow, Hassidim, and Lloyd first published their algorithm, experiments have been performed and improvements have been made to the versions they envisioned. Originally the intent of this work was to complete an implementation of HHL but through research it became clear that more could be gained from building on top of already existing work, as for both the HHL algorithm and the VQLS there are existing implementation.

3.1. IBM's Quantum network

As the race to build a Quantum Computer is ongoing so is the race to create the software they will run on. IBM has made it far on both fronts as they have built working Quantum computers and brought them online along with building a software platform based on the Open Quantum Assembly Language (OpenQASM)[7].Building on the OpenQASM, IBM has made an open-source framework for developing and test quantum algorithms called the Quantum Information Science Kit (Qiskit) and the cloud computing platform IBM Q Experience. Qiskit is a python package that allows high-level interface with the OpenQASM language and it can be run on and set up on your own machine like any other python package. Qiskit can let you run quantum simulations locally as well as providing a backend to IBM's own quantum -simulators and -computers [2]. As Qiskit is an open platform many are developing algorithms on it and so both HHL and VQLS have already been implemented. This implementation will be the basis of the next phase of experiments in this work. Using both the simulators and hardware available through the IBM Q Experience, and developing on the Qiskit modules the next part of this project.

Hardware

Quantum computers are not really here yet. Thought the effort to develop them is growing and new publications show improvements in this field every month the truth is we are not working with perfect qubits. What we are working with is noisy and prone to errors, and very small scale systems, on the order of 10 qubits, these systems we call noisy intermediary scale quantum computers or NISQ for short. NISQ systems are good for testing small scale circuits that take a short time to run but are not taking over from classical computers yet. Though as better techniques develop it may very well be that NISQ quantum computers are all we need to reach quantum advantage in some cases [14]. Gate models used by IBM are the CNOT gate, Identity and the rotational gates U_1, U_2, and U_3, this gate set is universal.

Simulators

While we are developing better and faster hardware the fastest and most reliable way to test quantum algorithms is by using a classical simulations. As stated by the Church-Turing thesis all Turing machines can compute the same functions so any classical computer, that is Turing complete, can simulate a quantum computer. Qiskit gives a module that turns any computer running it into a quantum simulator, this gives the user a great advantage in quick learning to use and test quantum algorithms. As a part of this local simulator, it is also possible to run "noisy" tests, that is trying to simulate the real

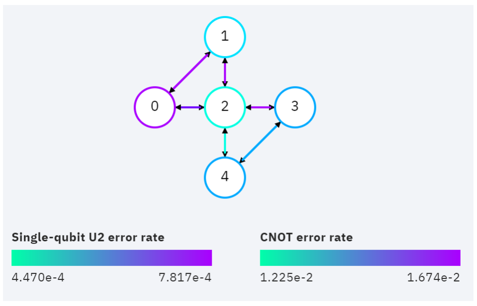

Figure 3.1: The ibmqx2 5 qubit machine, one of the machines available through the IBM Q experience.

hardware we are using today. This will be important in the continuation of the development of quantum algorithms such as the VQLS as they are designed for noisy hardware and should be tested on such. This implementation could also prove useful if tests on hardware are proving unsuccessful as the noise in the simulations can be calibrated, unlike in the hardware devices. Simulations also offer a greater size as IBM's online platform offers a 32-qubit simulator, which is double the number of qubits from the hardware offered.

3.2. A Practical Problem

To see how both implementations work at the current state of technology and to theories about their future a practical problem needs to be implemented to test their real use. Because the quantum formulation of the linear solver needs to be part of a bigger quantum problem to be truly advantageous as was mentioned when we discussed the formulation in chapter 2. This problem needs to be small enough in scale to be workable on quantum hardware and still be of practical value. The Poisson equation, setup in 1D is a good candidate for this job and a proper formulation using this classic problem will be the next step in this work.

Before trying a practical problem though a good test of the behavior of both implementations for different conditions of A needs to be carried out. This is to see the direct behavior of the two given certain conditions so that a better assessment can be made of the performance in the practical problem.

4 Thesis questions

For my thesis work want to explore further the functionality of the Quantum linear solvers I have introduced, using the implementation of the IBM Qiskit platform. I will test these implementations in both simulations and on real hardware. These tests will focus on practical matrices such as from the 1D Poisson equation, and testing varying- density and condition numbers. This is in part to explore in what directions the development of quantum algorithms for linear systems should take.

1. What approaches are there to solve linear systems on Quantum Computers?
2. How do these implementations scale with problems?
3. Can you integrate these solutions into a practical problem?
4. Can these methods get to quantum advantage in the near term?

Bibliography

[1] Dorit Aharonov. *World Scientific*, feb 2008.

[2] Abraham Asfaw, Luciano Bello, Yael Ben-Haim, Sergey Bravyi, Lauren Capelluto, Almudena Carrera Vazquez, Jay Gambetta, Shelly Garion, Leron Gil, Salvador De La Puente Gonzalez, David McKay, Zlatko Minev, Paul Nation, Anna Phan, Arthur Rattew, Javad Shabani, John Smolin, Kristan Temme, Madeleine Tod, and James Wootton. Learn quantum computation using qiskit, 2019. URL `http://community.qiskit.org/textbook`.

[3] Carlos Bravo-Prieto, Ryan LaRose, M. Cerezo, Yigit Subasi, Lukasz Cincio, and Patrick J. Coles. Variational Quantum Linear Solver: A Hybrid Algorithm for Linear Systems. sep 2019. URL `http://arxiv.org/abs/1909.05820`.

[4] X. D. Cai, Christian Weedbrook, Z. E. Su, M. C. Chen, Mile Gu, M. J. Zhu, L. Li, N. L. Liu, Chao-Yang Lu, and Jian-Wei Pan. Experimental Quantum Computing to Solve Systems of Linear Equations. feb 2013. doi: 10.1103/PhysRevLett.110.230501. URL `http://arxiv.org/abs/1302.4310http://dx.doi.org/10.1103/PhysRevLett.110.230501`.

[5] Yudong Cao, Anmer Daskin, Steven Frankel, and Sabre Kais. Quantum Circuit Design for Solving Linear Systems of Equations. oct 2011. doi: 10.1080/00268976.2012.668289. URL `http://arxiv.org/abs/1110.2232http://dx.doi.org/10.1080/00268976.2012.668289`.

[6] Quantum Computing. *Quantum computing: a gentle introduction*, volume 49. 2011. ISBN 9780262015066. doi: 10.5860/choice.49-0911.

[7] Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. Open Quantum Assembly Language 1 Background. Technical report, 2017.

[8] D. Deutsch and R. Jozsa. Rapid Solution of Problems by Quantum Computation. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 439(1907):553–558, dec 1992. ISSN 1364-5021. doi: 10.1098/rspa.1992.0167.

[9] Richard Feynman. The Feynman Lectures on Physics, 1963. URL `https://www.feynmanlectures.caltech.edu/`.

[10] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for solving linear systems of equations. nov 2009. doi: 10.1103/PhysRevLett.103.150502. URL `http://arxiv.org/abs/0811.3171http://dx.doi.org/10.1103/PhysRevLett.103.150502`.

[11] Dirk Meijer. The universe as a cyclic organized information system. john wheeler's world revisited. *NeuroQuantology*, vol 13:pp 57–78,, 03 2015.

[12] Memetician. A different kind of string theory: Antoni Gaudi - memetician — LiveJournal, 2007. URL `https://memetician.livejournal.com/201202.html`.

[13] Jian Pan, Yudong Cao, Xiwei Yao, Zhaokai Li, Chenyong Ju, Xinhua Peng, Sabre Kais, and Jiangfeng Du. Experimental realization of quantum algorithm for solving linear systems of equations. feb 2011. doi: 10.1103/PhysRevA.89.022313. URL `http://arxiv.org/abs/1302.1946http://dx.doi.org/10.1103/PhysRevA.89.022313`.

[14] John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, 08 2018. ISSN 2521-327X. doi: 10.22331/q-2018-08-06-79. URL `https://doi.org/10.22331/q-2018-08-06-79`.

[15] Benjamin Schumacher. *The Science of Information From Language to Black Holes*. The Great Courses, 2015. ISBN 9780198520115.