UNIVERSITY OF CALIFORNIA, MERCED

Classical and Quantum Resource Analysis for the Quantum Linear Systems Algorithm

A thesis submitted in partial satisfaction of the requirements
for the Master of Science Degree

in

Physics and Chemistry

by

Jon M. Inouye

Thesis Committee:

Professor Lin Tian, Chairperson
Professor Raymond Chiao
Professor Jay Sharping
Professor Jian-Qiao Sun

July 2010

The thesis of Jon M. Inouye is approved:

Raymond Chiao, PhD

Jay Sharping, PhD

Jian-Qiao Sun, PhD

Lin Tian, PhD
Chair

University of California, Merced

2010

ACKNOWLEDGEMENTS

I would like to express my appreciation to Lin Tian, my advisor, for her patient guidance; to Ann Kelley and Jay Sharping for introducing me to U.C. Merced. Department Chairs Sai Ghosh and Linda Hirst provided a relaxed learning environment.

Thanks to Kevin Mitchell, Professor of Physics, for excellent training in graduate quantum physics…

To the legendary Raymond Chiao, for giving cutting-edge modern physics the feel of that old-time yet so true experimental physics. Thanks are also extended to committee member Jian-Qiao Sun for his willingness to serve on my committee and for his excellent instruction in advanced classical dynamics…

To Chancellor Steve Kang, for taking a genuine interest in students at all stages of development…

Finally, I would like to acknowledge Carrie King and De Acker for their encouragement during moments of stress, and to the tight-knit community of physics graduate students and Natural Science staff for their support.

I dedicate this thesis to Mitsuo Inouye, M.D.

Noted Physician and Educator

My Father

1925-2007

Graduate of U.C. Berkeley (Biochemistry)

and

U.C. San Francisco (Medicine)

He taught me to believe in myself and to never give up.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT OF THE THESIS

Classical and Quantum Resource Analysis for the Quantum Linear Systems Algorithm

by

Jon M. Inouye
Master of Science in Physics and Chemistry
University of California, Merced, 2010
Professor Lin Tian

Recently (2009) a quantum algorithm for solving a system of linear equations has been proposed. The algorithm by Harrow, Hassidim, and Lloyd has attracted considerable attention in the quantum algorithms community, due to the broad potential applications of a rapid linear equations solver. The contribution of this thesis is to analyze the classical and quantum resources required for implementation.

The thesis has two major tasks. We first survey the field of quantum algorithms. The papers which established this field (e.g., Feynman and Deutsch) to recent works are reviewed. Different classes of quantum algorithms are examined, including those based on the Fourier transform, quantum searching, and quantum walk.

For the second task, we focus on the algorithm by Harrow, Hassidim, and Lloyd. The advantages of the algorithm are an exponential performance gain over classical algorithms (under conditions of sparse operator matrices and few selected measurements from the solution set), and fewer data registers.

In the second part of the thesis, we study the classical resources required for implementation of the algorithm. Since classical resources can determine the ultimate efficiency of the quantum algorithm, the optimal use of classical resources is mandatory.

We demonstrate how a classical computer may handle certain computations (e.g., time evolution) and feed these into the quantum circuit implementing the algorithm. Thus, the classical and quantum resources for implementing the algorithm are described. Through a detailed analysis of the classical resources, we hope to understand how these resources may be optimized. This work can therefore contribute to the design of efficient quantum algorithms.

Chapter 1: Introduction

This thesis first surveys the field of quantum algorithms, from inception to recent works. We then focus on the recent quantum algorithm by Harrow, Hassidim, and Lloyd [HHL09a]. Our goal is to describe the classical (and related quantum) computer resources necessary to implement the Harrow-Hassidim-Lloyd algorithm for solving linear equations.

Classical computation is performed using physical circuits that behave according to the laws of classical physics. Thus, classical computer algorithms assume that the steps of each algorithm will be executed on a classical computer (although, as we will see later, classical computer algorithms can also be executed on what is known as a quantum computer).

The digital gates of a classical circuit are Boolean, meaning that the inputs and outputs are in definite states (either 0 or 1). The results (outputs) of a digital gate – and hence a classical computation – are mostly irreversible, meaning that we cannot redo or retrace the computation to its original input. The AND and OR gates, for example, are not reversible, although the NOT gate is indeed reversible.

However, classical computers are not limited to electronic gates. Fredkin and Toffoli [FT82] modeled a classical computer model based on the perfectly elastic collisions of billiard balls. Paun created a classical computing model based on the logical operations of cell membranes [Pau03]. Like classical mechanics, a classical computer is deterministic in nature, meaning that every input determines a

predictable, definite output. No ambiguity or uncertainty exists about the state of a digital gate.

Alan Turing [Tur36] invented a theoretical model of classical computing which was independent of physical implementation. The model consisted of a stored program, a set of finite states (including the start state, q_s, and ending state, q_h), a read/write head, and a tape. The read/write head could write or read symbols from the tape, one cell at a time. The tape could be moved one cell at a time (to the left or to the right). All the symbols written or read from the tape came from a finite alphabet defined for the machine. The program lines controlling the operation of the model were represented by tuples of the form, <q, x, q', x', s>, where q was the current state of the system, x was a symbol read from the tape, q' was the next state of the system, x' was the symbol written onto the tape, and s represented whether the read/write head moved leftward to the previous cell (L), advanced right to the next cell (R), or remained stationary at the current cell on the tape (0).

This model later became known as a classical Turing Machine (TM). The simulation of any classical algorithm could then be followed using the TM; the behavior of the algorithm could be modeled exactly. Turing's motivation for the model was not to design the fastest or most efficient computer to implement the algorithm. Rather, the TM was used to describe what can and what cannot be computed.

The Church-Turing thesis asserts that any algorithm that is computable can be simulated on the Turing machine model. An algorithm that is not computable (such as the "halting problem") cannot be simulated on the TM. More formally:

> The class of functions computable by a Turing Machine corresponds exactly to the class of functions which we would naturally regard as being computable by an algorithm. [NC00].

The Church-Turing Thesis has not been formally proved, because of the ambiguity of what is meant by the phrase, "naturally regard as being computable." For the moment, the thesis is an intuitive rule-of-thumb for what we consider to be computable or not.

No classical device operating according to classical physics is more powerful than a Turing Machine (i.e., given enough time the TM can simulate any classical system). However, a device based on quantum mechanics could be more powerful than a device based on classical mechanics. The quantum equivalent to a Turing Machine is the Quantum Turing Machine (QTM). More generally, a Quantum Turing Machine is more powerful than the classical Turing Machine [BV93]. The quantum read/write head could write quantum states (a superposition of several states) to the tape instead of single symbols. The transition between quantum states, as described by the program lines (tuples), could also include a probability amplitude between states. The quantum TM uses qubits (quantum bits) instead of bits. Qubits are not distinct and exist in the quantum mechanical world as a superposition of states.

A qubit has corresponding probability amplitudes for its possible states ($|0\rangle$ or $|1\rangle$) and hence is probabilistic (with complex vectors described in Hilbert space) rather than deterministic in nature. In particular, suppose we have a qubit called q_0. Then:

$$q_0 = (c_1)\,|0\rangle + (c_2)\,|1\rangle \qquad (1.1)$$

The standard operations of a quantum computer are to initialize the superposition of states, to perform operations on qubits (such as the Hadamard transform), and to measure qubits for the answer. The operations of the QTM are always unitary. When measuring the qubit with the amplitudes described above (c_1 for state $|0\rangle$ and c_2 for state $|1\rangle$), we may randomly get either a binary digit of 0 or 1. The probability of digit 0 is $|c_1c_1^*|$. Digit 1 has a probability of $|c_2c_2^*|$.

In contrast to a classical computer, a quantum computer is always reversible. This feature (reversibility) can be exploited in rapidly performing an inverse Fourier transform. As we will elaborate further in our literature review, a quantized Fourier transform can be used to rapidly estimate the phase of waveforms that meet certain criteria, answering certain types of problems that depend on phase (such as the phase of an eigenvalue).

Quantum states are not only in superposition, they are also entangled with one another. In a sense the entangled states can “read” each other. By exploiting the superposition of states and quantum entanglement, we can achieve a form of massive parallel processing [Deu85]. Certain quantum algorithms have been discovered that can achieve exponential performance improvement over classical algorithms (most significantly, Shor’s algorithm to perform factorization of integers). Other quantum algorithms have achieved polynomial speedup (e.g., Grover’s search algorithm).

The primary interest in quantum algorithms today is to find new algorithms that make certain problems tractable that are intractable using classical computers.

The implications for fields such as cryptography, computer networks, and database searching are profound if a practical quantum computer could be built.

We mentioned some of the advantages of quantum computers, i.e., that they can achieve massive parallelism. But for quantum computers to realize these benefits, certain difficulties must be overcome. Quantum states are difficult to maintain due to interference from the outside environment. There is the ongoing challenge of dealing with noise in constructing an actual quantum computer.

Furthermore, algorithms that are not computable (or undecidable) on a Turing machine are also not computable on a quantum Turing machine. There are problems that are equally difficult on a classical as well as quantum computer. If we cannot find ways to exploit the quantum parallelism in a problem (i.e., if the problem is more sequential in nature and does not lend itself to being decomposed into parallel pieces), then running an algorithm on a quantum computer does not offer a clear advantage. For example, an order processing system for a retail store is highly dependent on real-time, sequential transactions; a quantum computer to implement the order processing system may not offer any clear benefit.

Thus far (2010), no practical quantum computer has been built. Understanding the resources required by quantum algorithms may also provide insight to the actual requirements of a practical quantum computer.

For instance, some parts of a quantum algorithm may operate using classical computer resources, while other portions of the algorithm may operate using quantum circuits. This thesis studies in detail one such quantum algorithm and

explores how certain steps may be fulfilled classically, while other steps of the algorithm are best fulfilled using quantum circuits.

Let us begin by surveying the seminal papers in the field. The following literature review is not exhaustive. Rather, the literature review describes those key papers that contributed to the development of the field of quantum algorithms.

Chapter 2: Literature Review

2.1 Foundations

The original motivation for quantum computing was to simulate physical quantum systems. Feynman [Fey82] pointed out the inherent difficulty of simulating quantum mechanics using classical computers. As the number of particles in a quantum system grows (i.e., the number of quantum variables representing the system expands), the number of possible states increases exponentially. The processing power of classical computers would be insufficient to simulate this exponential growth. Feynman suggested that a computer based on the principles of quantum mechanics itself could successfully simulate quantum systems. Feynman even asserted, in what later became known as the Feynman Conjecture, that only computer systems based on quantum mechanics could successfully simulate quantum mechanics.

Seth Lloyd showed in a 1996 paper that the Feynman Conjecture was correct [Llo96]. Lloyd first pointed out that quantum simulation using a quantum computer induces interactions between physical quantum variables. Lloyd contrasted the quantum simulator with the classical computer simulations of the physical quantum system; as the size of the quantum system grows (i.e., the number of quantum variables is added), the classical computer cannot compensate for exponential growth in the number of states. However, since a quantum computer mimics the interactions of the physical quantum system using actual quantum variables, the growth in time

complexity (from the viewpoint of the quantum computer) is only linear growth. Thus, only computer systems using quantum mechanics can successfully simulate quantum mechanics.

In addition to the efficacy of the quantum computer as a simulator for quantum systems, an inevitable question began to emerge: Would it be possible to use a quantum computer to solve problems not possible (i.e., intractable in terms of time complexity) on a classical computer?

In a landmark work, David Deutsch [Deu85] proposed the first quantum algorithm with a demonstrated performance improvement over a classical algorithm. In fact, the quantum algorithm was exponentially more efficient than a classical algorithm. To support his analysis, Deutsch also proposed the Quantum Turing Machine, a quantum mechanics equivalent of the classical Turing Machine model.

Deutsch posed a problem with a one-qubit solution. As an illustration of "Deutsch's problem", consider the outstanding example given by Nielsen and Chuang [NC00]:

Suppose there are two people (Alice in Boston, Bob in Los Angeles) who are in two-way comunication. Alice sends Bob a number, x, which ranges from 0 to 2^n-1. Bob evaluates f(x), a function with an n-bit domain and one-bit range, which may return 0 or 1. Bob sends f(x) to Alice. It is assumed by both parties that f(x) is either constant for any value of x, or f(x) is balanced (contains an equal number of 0's and 1's). Alice must query Bob to determine whether f(x) is constant or balanced. In the worst case, Alice must query Bob $2^n/2 + 1$ times.

Deutsch's algorithm is implemented using the quantum circuit in Figure 1. The gate labeled "U_f" represents a "black box" that implements a unitary transformation from state $|x, y\rangle$ to state $|x, y \oplus f(x)\rangle$. The inputs are on the left of the gate, consisting of a Hadamard gate acting on state $|0\rangle$ for x. Thus the input for x is a superposition of states $|0\rangle$ and $|1\rangle$. The input for y is the state $|0\rangle$, making the output for y just f(x). The resulting state is $|\psi\rangle$:

$$|\psi\rangle = (|0, f(0)\rangle + |1, f(1)\rangle) / \sqrt{2} \qquad (2.1)$$

The resulting state contains information on both f(0) and f(1). The Deutsch algorithm makes use of quantum parallelism: a function, f(x), could be evaluated for different values of x simultaneously. However, it should be noted that quantum parallelism differs fundamentally from the classical parallelism that can be implemented on classical computers (e.g., if we have several classical computers operating in tandem). In quantum parallelism, two different alternatives (the different values of x in function f(x)) may interfere with one another to provide a global property of the function. In classical parallelism, the two alternatives do not interfere and exclude one another.

Using the quantum circuit, only one query is necessary, thus providing an exponential speedup over a classical algorithm.

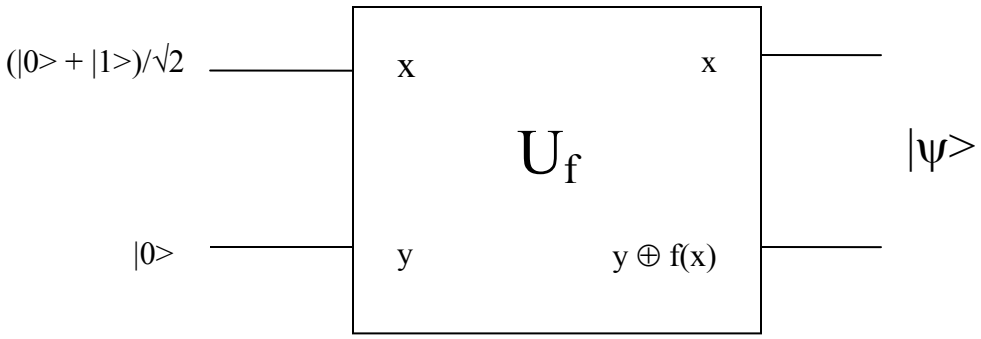

Figure 1. Quantum Circuit for Deutsch Algorithm

The original Deutsch algorithm was generalized from 1-bit to n-bits by Deutsch and Jozsa in 1992, to become known as the Deutsch-Jozsa Algorithm [DJ92]. The Deutsch-Jozsa algorithm led to a "gold rush" for other algorithms that could be superior to classical algorithms. Thus, the papers by Feynman and Deutsch are generally attributed to starting the entire field of quantum computing [Mer07]. However, the Deutsch-Jozsa algorithm itself is rather contrived; it has no known applications, serving only to demonstrate that exponential speedup is possible using a quantum computer.

The implications of the original Deutsch paper were further explored by Berthiaume and Brassard in two papers. "The Quantum Challenge to Structural Complexity Theory" [BB92a] proved mathematically that there are certain problems in which a quantum computer could solve in polynomial time, while a classical computer would require exponential time. "Oracle Quantum Computing" [BB92b] showed that there existed entities called *quantum oracles*, which could be used to

model quantum algorithms for polynomial time solutions to classically exponential-time problems. Quantum oracles could be viewed as "black boxes" which perform unitary transformations, from states such as $|0\rangle |x\rangle$ to states of the form $|f(x)\rangle |x\rangle$. Quantum oracles have been described as analogous to the subroutines of classical computer programs; quantum algorithms call quantum oracles during the processing of quantum states in the same sense that classical computer programs call subroutines to perform specific actions. However, unlike classical subroutines, the quantum oracle has the requirement that the invocation must cost only unit time and that the quantum oracle must not leave behind any garbage (i.e., excess bits) beyond the computed answer.

2.2 Models for Computation: Quantum Turing Machines and Quantum Circuits

Significantly, Bernstein and Vazirani showed that there exists an efficient, universal quantum Turing machine [BV93]. The authors proved that such a universal quantum Turing machine is more powerful than a classical Turing machine. Since the QTM is more powerful than a classical TM, then we may also logically conclude that a classical algorithm can be run on a QTM, although the converse is not true. That is, we cannot run a quantum algorithm on a classical computer.

Just as a quantum algorithm can be computed on a QTM, a quantum algorithm can also be implemented as a quantum circuit, as we saw in the example of the Deutsch-Jozsa algorithm.

Yao [Yao93] and Deutsch [Deu89] were largely responsible for the quantum circuit model being used as the basis for modeling quantum algorithms. Modern quantum algorithms are typically described as low-level iterations acting on qubits entangled according to the appropriate quantum circuit model. Yao proved that the quantum circuit model is equivalent in power to a quantum Turing machine. It was also shown in the paper, "Elementary Gates for Quantum Computation" [BBCD^{+}95] that a set of gates consisting of a one-bit quantum gate and a two-bit exclusive-OR gate could serve as a universal gate.

Figure 2 shows the basic quantum gates that will be used in quantum circuits (e.g., a quantum Fourier circuit).

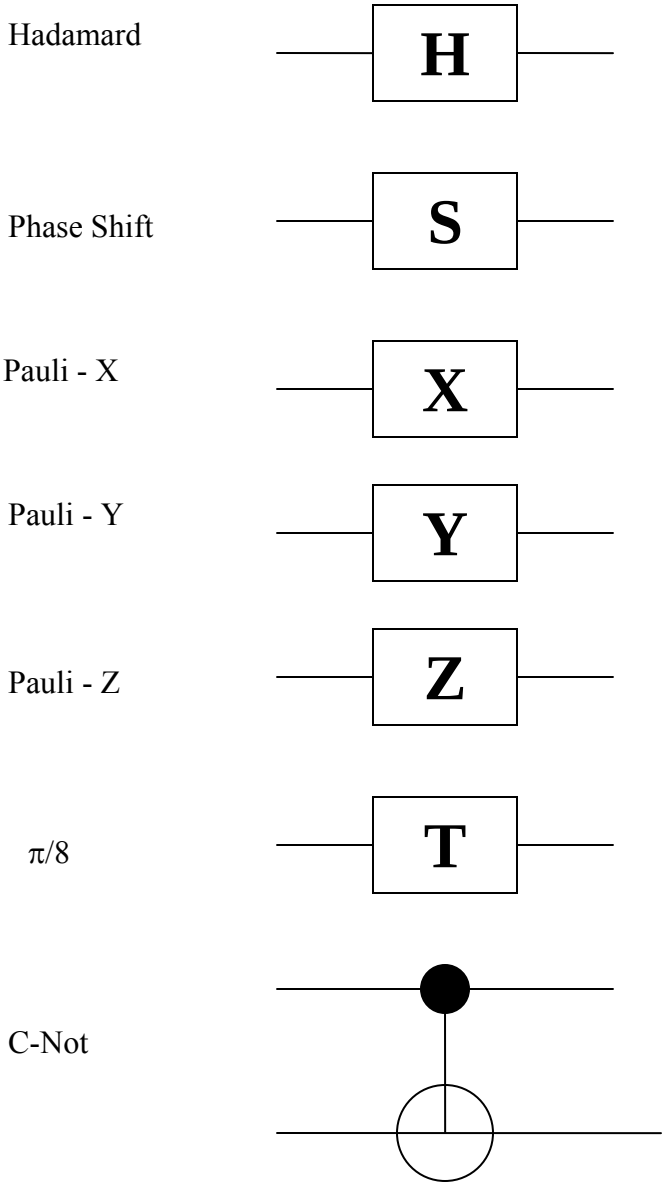

Figure 2. Typical Quantum Gate Notations

The gates are read from left to right, where input occurs on the line at the left and output results on the line to the right. The Hadamard gate implements a Hadamard transform on a single qubit. The Phase Shift, S, implements a shift in phase on a single qubit. Pauli-X, Pauli-Y, and Pauli-Z implement rotations on a single qubit about the X, Y, and Z axis respectively. The $\pi/8$ gate, T, actually implements a shift of $\pi/4$ (the name is a historical misnomer that has managed to persist).

The C-Not gate, or Controlled-Not, operates on two qubits. One of the qubits acts as a control, the other serves as data. If the control is set to "ON" then the operator acts on the data qubit, performing a NOT logical operation. If the control is not set to "ON", then the operator leaves the data qubit alone.

Figure 3 lists the corresponding matrix representations for the gates given in Figure 2.

Hadamard	H =	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$
Shift	S =	$\begin{bmatrix}1 & 0\\ 0 & i\end{bmatrix}$
Pauli-X	X =	$\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$
Pauli-Y	Y =	$\begin{bmatrix}0 & -i\\ i & 0\end{bmatrix}$
Pauli-Z	Z =	$\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$
π/8	T =	$\begin{bmatrix}1 & 0\\ 0 & \exp(i\pi/4)\end{bmatrix}$
C-Not		$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$

Figure 3. Matrix Representations of Quantum Gates

Like the classical Turing Machine, the behavior of the Quantum Turing Machine is represented as a series of tuples containing the mapping between quantum states, the alphabet, the movement left or right on the quantum tape, etc. Unlike the classical TM, probability amplitudes are assigned to the state transitions and what is read or output during each transition. The behavior of the quantum computing device is probabilistic rather than deterministic in nature. However, the quantum circuit

model is frequently used instead of the QTM, since the circuit model is more convenient and more visual in nature than the (perhaps) more unwieldy QTM representation. In this thesis we focus on the quantum circuit model.

Charles Bennett [BBBV97] summarized both the strengths and weaknesses of quantum computing, and tried to establish the upper bounds on time complexity for QTMs relative to classical Turing Machines. In light of the discovery that certain quantum algorithms could achieve exponential speedup over classical algorithms, the question was asked whether quantum computers could solve problems in the class of NP in polynomial time. Bennett and his co-authors showed that a quantum computer could not solve problems in the NP-class in $O(2^{n/2})$ time. What was established was that there is no black-box technique for solving NP-complete problems in polynomial time merely by using quantum mechanical features of the QTM.

Dan Simon [Sim94] pointed out that a Hadamard transformation (a special case of the more general Fourier transform) could be used to find the hidden period of a function. In the problem described by Simon, we first wish to determine whether a function, called f, could be invariant under an exclusive-or mask; if the function is not invariant under exclusive-or, then we wish to find a nontrivial period s, such that:

$$f(x \oplus s) = f(x) \qquad (2.2)$$

where $\oplus$ is the "exclusive-or" operation. We assume that each string x is n bits in length.

By applying the Hadamard transformation repeatedly and in sequence, we obtain a quantum superposition of all possible binary strings. In the case where f is invariant, s will be a random string. We can also manage the case where f is not invariant and a nontrivial period exists. A simple evaluation of $f(0^n)$ and f(s*) will determine whether s is a random string or whether s is the true period.

Simon's algorithm achieved linear time complexity, O(n), compared to the exponential growth of its classical counterparts. While the Deutsch-Josza algorithm achieved exponential speedup but had no known applications, the Simon algorithm had far more applicability, especially to problems where order-finding was central to the problem. In fact, Simon's algorithm formed the inspiration for Peter Shor's seminal factorization algorithm.

2.3 Shor's Factorization and Discrete Log Algorithm

Shor [Sho94] proved that well-known problems such as prime number factorization and discrete log estimation were reducible to order-finding (of the type solved by Simon's quantum algorithm). That is, given a large integer N and an arbitrary positive integer, x, such that $x < N$, the order r is defined as the least positive integer such that $x^r \bmod N = 1$. The integer r is also referred to as the "order" of x mod N.

Shor's algorithm used Fourier transforms instead of simpler Hadamard transformations to determine the order. An input quantum register was loaded with integer candidates for r. An output quantum register was loaded with the unitary transformation on r ($x^r \bmod N$). The Fourier transform was then applied to the input

quantum register, followed by an inverse Fourier transform and measurement of the input quantum register. The result was an estimated phase value, from which order r could be approximated. The order r could then be incorporated into a gcd (greatest common divisor) calculation between r and N, performed classically, to obtain the factor. A more complete description of Shor's algorithm for factorization is given in Chapter 3.

Performing factorization and discrete log estimation were considered to be classically hard problems. In fact, the very intractability of integer factorization was the basis for cryptographic algorithms such as RSA. However, Shor demonstrated that by using quantum order-finding, factorization and discrete log estimation could be performed in polynomial time.

The implications of this result were highly significant. If a quantum computer could be built, Shor's factorization algorithm would render the current generation of cryptographic systems based on RSA obsolete, since any RSA-encrypted code could be deciphered in polynomial time.

Ekert and Jozsa [EJ96] provided a lucid and highly accessible description of Shor's algorithm. They discussed the significance of the algorithm both within the contexts of computer science and physics, and suggested experiments that would contribute to the implementation of the algorithm. The Ekert and Jozsa paper is therefore a recommended companion to the original, groundbreaking work by Peter Shor.

"Efficient Networks for Quantum Factoring" [BCDP96] analyzes the memory and time complexity of a hypothetical quantum computer running the Shor

factorization algorithm. Issues such as garbage collection to preserve memory space, and the implementation of quantum gates using the ion trap technique, are also discussed. The paper is notable in that the authors study which classical and quantum resources would be required for the Shor algorithm; the authors also discuss how some of the classical and quantum resources interrelate (for example, a classical subroutine would calculate a new factor using the result from a quantum oracle which determined the order).

Kitaev [Kit95] generalized Shor's results for factorization and the discrete logarithm. He proved that both factorization and discrete logarithm were special cases of what is called the "Abelian Stabilizer Problem" (ASP). The ASP can be described as follows:

We are given a "black box" function called F, which can operate over an arbitrary, finite Abelian group. Let H = any Abelian group, $s \in H$, and $x \in$ any finite set. We are also given that F(s, x) = x, defining the function as periodic. In the stabilizer problem, we wish to find the "stabilizing" element, s, such that:

$$F(gh, x) = F(g, f(h,x)) \text{ and} \qquad (2.3)$$
$$F(gs, x) = F(g, x).$$

It is evident from the above that the action of finding the order, and hence factorization and discrete logarithm, are indeed included as instances of ASP.

Kitaev invented a phase estimation algorithm which broadened the scope of Shor's original algorithm, allowing the algorithm to perform a Fourier transform over an arbitrary, finite Abelian group.

2.4 Grover's Search Algorithm

In the late 1990s, Lov Grover discovered a way to use quantum mechanics to help in searching for data [Gro97]. In Grover's scheme, the search items were represented in a quantum system as a superposition of states, with the same probability amplitude for each search item (state). I.e., given a total of N search items, each state had an amplitude of $1/\sqrt{N}$. Rather than using a full-fledged Fourier transform, Grover used a simpler phase estimation technique (referred to as a "diffusion transformation" in the paper) to alter the phase until the correct item was obtained. Grover's work was significant, in that he demonstrated how quantum computing could be used for a highly pertinent real-world application.

A primary advantage of Grover's search algorithm was its relative simplicity: the technique can be considered more streamlined and less complicated than a full quantum Fourier transform. A comparative drawback, however, is that the search algorithm achieves only polynomial-time improvement rather than an exponential speedup. Given N random search items, a classical search algorithm requires, in the worst case, O(N) operations. Grover's search algorithm requires O($\sqrt{N}$) operations.

In the paper, "A Fast Quantum Mechanical Algorithm for Database Search" [Gro96], Grover extended the search algorithm to include searching on a database. We provide the details of Grover's algorithm in Chapter 4.

Brassard, Hoyer, Mosca, and Tapp [BHMT00] were able to generalize Grover's algorithm into a concept called "amplitude amplification." During each iteration of Grover's algorithm, the pattern of the algorithm was to increase the

probability amplitude of certain search criteria, thus increasing the likelihood of success in obtaining the search item. When the probability amplitude was increased to a very high value (perhaps close to 1), the final measurement would be performed.

In "Quantum Counting" [BHT98], the authors extended the amplitude amplification process to search algorithms possessing classical heuristics. Their refined amplitude amplification process was called "quantum counting", or alternately, "approximate counting".

2.5 Generalization of Quantum Algorithms: Hidden Subgroup Problems

Other authors noticed the overall patterns emerging from the now widely studied Shor factorization algorithm and Grover search algorithm. Cleve et. al [CEMM97] viewed quantum computation as multi-particle interference. The Shor and Grover algorithms were performing phase estimation from the superposition of waves in the quantum Fourier transform.

By the turn of the century, the field of quantum algorithms had advanced to the point where the first textbook was written on the topic [NC00]. In the early 2000s, quantum algorithms could be classified into three general types: those based on Shor's algorithm using the quantum Fourier transform, those based on Grover's quantum search algorithm, and those algorithms intended for quantum simulation.

The quantum algorithms based on Shor's algorithm fell into a more general category called solving the "hidden subgroup problem." [Sim94] [NC00]. Examples of algorithms that solve the hidden subgroup problem include Deutsch's algorithm,

the period-finding or order-finding algorithms, the discrete logarithm, and the Abelian stabilizer. More specifically, we define the hidden subgroup problem as follows:

Let K be a hidden subgroup within a larger, finite group, G. Let f be a function that maps finite group G to a finite set, X. That is, f: $G \rightarrow X$. Function f is constant and distinct on the cosets of K. Suppose there is an element $g \in G$, and an element $h \in X$. Given a unitary transform, $U |g\rangle |h\rangle = |g\rangle |h \oplus f(g)\rangle$, where "$\oplus$" is a binary operation on X, the hidden subgroup problem is to find a generating set for subgroup K.

2.6 Shor's Error Correcting Algorithm

While the theory of quantum algorithms appeared to be rapidly advancing, the hardware implementation of quantum computing was (and still is) beset by the problem of decoherence. This thesis will not concentrate on the schemes for hardware implementation. Although error-correcting codes are part of the implementation process, we will briefly mention a few significant papers.

Calderbank and Shor [CS96] showed that good error correcting codes for quantum computing can exist, setting the theoretical basis to overcome decoherence in quantum superpositions. In "Fault Tolerant Quantum Computing", Peter Shor [Sho96] introduced the first quantum error correcting codes, and showed how these codes could be used in a quantum representation without having to decode (measure) the information.

In the remainder of the Literature Review, we will discuss recent works (from 2002 to 2009) in the field of quantum computing.

2.7 Recent Works

What is noteworthy about the recent works has been the quest for alternative algorithms which are not based on the Fourier transform. I would like to mention two new types of algorithms among those that have appeared since 2002: those quantum algorithms based on the quantum walk and those algorithms based on adiabatic quantum computing.

The paper, "Exponential Speedup by Quantum Walk" [CCDF$^+$02] introduced the quantum walk. In classical thermodynamics, a random walk involves successive random steps, modeled by a Markov chain. The goal of the random walk is to model a trajectory that is moving at random (e.g., a molecule in a gas). A quantum walk is analogous to a random walk; the quantum walk can be modeled as a probability distribution. However, the entity that is performing the quantum walk (let us call this entity a "quantum walker") exists in a superposition of states. Also, the states of a quantum walk are defined in Hilbert space.

Let the states of a quantum walk represent vertices on a graph. The goal of a quantum walk algorithm would be to traverse the entire graph. The progress of the quantum algorithm would be based on queries to a "black box" (quantum oracle), asking about the local properties of a vertex. While a classical random walk algorithm would require exponential time complexity to traverse a graph, a quantum walk would take only linear time. As Childs et. al demonstrated, applications such as Grover's search algorithm could be modeled using a quantum walk.

In "Spatial Search by Quantum Walk" [CG04], the authors apply a continuous-time quantum walk algorithm for multidimensional database searching. A polynomial speedup of O($\sqrt{N}$) was obtained over classical probabilistic methods.

By 2009, the quantum walk had become one of the well-known quantum algorithms used for searching. Childs [Chi09b] showed how the discrete quantum walk could be used to estimate the continuous quantum walk, and thus how the discrete quantum walk could be used to simulate Hamiltonian dynamics.

The other type of quantum computing that has been recently proposed, adiabatic quantum computing, applies the "adiabatic theorem" of quantum mechanics to perform computations [FGGS00].

The adiabatic theorem states that if a system is in a particular eigenstate (e.g., the ground state), and the perturbation on the system is slow, then the system remains in that eigenstate. In addition to the slow perturbation on the system, adiabatic computing also requires that large band gaps exist between the eigenvalue (corresponding to the eigenstate) and the rest of the Hamiltonian spectrum.

We desire to evolve the Hamiltonian from an initial, simple Hamiltonian (set to the ground state), to a final, more complex Hamiltonian (which is still in the ground state). The final Hamiltonian represents the solution to a given computational problem.

Since the system is always in the ground state during evolution, adiabatic computing intends to get around the problem of decoherence. However, there are still problems with adiabatic computing -- outside interference could still tip the ground state into the first energy state, ruining the calculation.

Aharonov and Ta-Shma [AT03] focused on developing tools to generate quantum states by adiabatic evolution. The authors believed that the notation of Hamiltonians and spectral gaps was a "natural" way of expressing adiabatic quantum computing. By focusing on the process of quantum state generation, they believed that quantum computing itself could be better understood.

A recent paper by Gottesman and Irani [GI09] studied the time complexity of a class of problems that is invariant under spatial translation. An analogy was drawn between two related problems, one classical and the other quantum mechanical.

The classical tiling problem involves a set of m tiles, and rules specifying how adjacent tiles may be positioned. The input to the problem is an integer, N, indicating an N x N grid that must be tiled according to the rules. Given the integer N, the classical tiling algorithm must find the possible tiling of an r-dimensional grid.

The quantum problem must approximate the ground state energy of a quantum system when the Hamiltonian is invariant under spatial translation. The interactions between particles in the quantum system only occur between neighboring (adjacent) particles on an r-dimensional grid.

Thus, the problems are similar. In both the classical tiling problem and the quantum mechanical problem, the particles (or tiles) are adjacent, and the rules act as constraints on the possible states of the systems. The authors proved that the classical tiling problem is NEXP-Complete, and that the quantum problem is QMA_{exp}-Complete. (Note: QMA_{exp}-Complete is the quantum version of NP-Complete). Gottesman and Irani pointed out that if we could find an algorithm that could run in

polynomial time in N, then this would imply that EXP = NEXP and BQEXP = QMA_{exp}.

Within the past year, a quantum algorithm to solve a system of linear equations was proposed by Harrow, Hassidim, and Lloyd [HHL09a][HHL09b]. The quantum linear algorithm allows for exponential speedup over classical algorithms. Given a system of N linear equations, the algorithm requires O(log(N)) data registers, in contrast to O(N) for classical algorithms. Suppose that a linear system of equations has the form $A\vec{x} = \vec{b}$, where A is a Hermitian and unitary operator, and $\vec{x}$ and $\vec{b}$ are vectors. (The quantum linear algorithm represents $\vec{x}$ and $\vec{b}$ as quantum states in Hilbert space). If we are not interested in all values of $\vec{x}$, but only in some special feature of $\vec{x}$, such as a particular expectation value, then in the best case the quantum linear algorithm may achieve an exponential speedup over the classical algorithm.

The quantum linear algorithm first prepares the quantum states for |b>, using the technique of Grover and Rudolph [GR02]. Eigenvector |b> is decomposed and transformed into the basis states for the operator, A. Then a conditional Hamiltonian time evolution operator is applied to |b>, where the unitary operator, A, is a sparse matrix. We use the techniques of Hamiltonian simulation [BACS06] to apply operator A to the eigenvector |b>. Then the phase estimation algorithm is applied, using the discrete Fourier transform to obtain a multi-qubit estimation for the phase of the eigenvalue for A. We add a qubit and rotate about the basis state established by the phase, measure the last qubit, and determine the inverse phase conditioned on the state, |1>, of the added qubit. The result of the inverse phase is our desired solution set:

$$|x\rangle = A^{-1}|b\rangle \qquad (2.4)$$

Until the eigenstates for $|x\rangle$ are measured, however, the solutions will be unknown. As indicated above, we must measure only certain properties of $|x\rangle$, rather than all values of $|x\rangle$, otherwise the quantum linear algorithm offers no performance gain over the classical algorithms for solving linear systems.

Andrew Childs [Chi09a] pointed out some of the implementation problems to the quantum linear system algorithm. The preparation for the states in eigenvector $|b\rangle$ must be quick. If the data for $|b\rangle$ is given explicitly in terms of classical data, then preparation may not be rapid. The Hermitian operator, A, must be sparse, as measured by a small condition number, κ. If the matrix for A is non-sparse, the algorithm loses its advantage over classical methods. Although the potential applications of a quantum linear system algorithm are broad, no specific task has yet been proposed (as of this writing in mid-2010) for which the quantum linear system algorithm of Harrow-Hassidim-Lloyd would outperform classical linear equation solvers.

2.8 Summary of Literature Review

In summary, we have discussed the seminal papers which defined the field of quantum algorithms. The algorithms which use the quantum Fourier transform to perform phase estimation form a broad category of quantum algorithms to solve what was called the "hidden subgroup problem." These algorithms included the Deutsch algorithm, the Shor factorization and discrete log algorithms, order-finding, period-finding, and the Abelian Stabilizer. The other types of algorithms were based on Grover's search algorithm, which do not use the Fourier transform, but rather use techniques intended to increase the probability amplitude of selected states, thus converging to a solution. We mentioned recent algorithms (2002-2009) which were not based on the hidden subgroup problem: the quantum walk, which was based on quantum oracle queries over a quantum graph; and the adiabatic quantum computer, which performs computation by taking advantage of the adiabatic theorem of quantum mechanics. We briefly mentioned quantum simulation, and error-correcting codes for quantum computing.

Finally, we discussed a significant recent algorithm (by Harrow, Hassidim, and Lloyd), which uses quantum computing to solve a system of linear equations. The quantum linear system algorithm is a phase estimation method. The algorithm can achieve polynomial and even exponential speedup over classical computing under certain conditions (i.e., sparse operator matrices, and measurement of final quantum states performed only locally). We also discussed some of the implementation problems of the algorithm.

In the remainder of the thesis, we will discuss in detail Shor's factorization algorithm and Grover's search algorithm. We will also cover the quantum linear system algorithm, and we will perform an analysis on the classical and quantum resources needed to implement the algorithm.

Chapter 3: Shor's Factorization Algorithm

3.1 Introduction to the Algorithm

In this chapter, we will provide a detailed description of Shor's Factorization Algorithm [Sho94].

Given an integer, N, a prime factorization algorithm decomposes the integer into a product of prime numbers. Classical prime factorization algorithms require exponential time complexity to factor large integers. In fact, the security of an encryption algorithm called RSA [RSA78] is based on the hardness of factorizing integers. However, Shor's factorization algorithm exploits the massive parallelism of quantum computing and can factorize large integers in polynomial time. The actualization of a practical quantum computer therefore has serious implications for cryptography as well as other areas of science.

Suppose we desire to factor a positive integer, N. We are given a randomly selected number, x, such that $x < N$. It is trivial for us to check if x and N have a common factor by using Euclid's algorithm, prior to running our quantum algorithm. If x and N have a common factor, then we have found a factor of N and are done. However, for purposes of demonstration, let us assume that x and N do not have a common factor. Then we desire to find the least integer, $r > 0$, such that:

$$x^r \bmod N = 1 \qquad (3.1)$$

This least integer, r, is called the "order" of x mod N. Shor proved that the factorization problem is reducible to finding the order of x mod N. Once r has been determined, we can find the factor through using the classical greatest common divisor algorithm.

The input to the quantum algorithm is the classical integer, N. The output is a prime number factor to N. Two quantum registers are used to represent binary integers; the first register is used as input, the second register is used as output. Also required is a scratchpad used as temporary workspace. The workspace is cleared after each subroutine of the algorithm.

We determine a constant number, q, which is a power of 2 such that $N^2 \leq q < 2N^2$. We also choose an arbitrary positive integer x, such that $x < N$. Thus, N, x, and q will be constant throughout the algorithm. Let the number, a, represent the order of x mod N.

With the order-finding algorithm serving as the core, we describe the steps of Shor's factorization algorithm as follows.

3.2 Steps of the Algorithm

Step Zero: Preprocessing. We first check to see if a factor can be trivially determined.

If N is even, return the number 2.

Use Euclid's algorithm to check if x, N have a common factor.
If yes, return the common factor.

Otherwise, proceed to Step One.

Step One: Initialize the first register with the superposition of states labeled by a, and clear the second register.

$$(1/q^{1/2}) \sum_{a=0}^{q-1} | a > |0> \qquad (3.2)$$

Step Two: Compute x^a (mod N) into the second register. Leave |a> in the first register (so the computation is reversible):

$$(1/q^{1/2}) \sum_{a=0}^{q-1} | a > |x^a \text{ (mod N)}> \qquad (3.3)$$

Step Three: Apply the Fourier Transform to the first register, mapping states |a> to |c>:

$$|a> \; = \; (1/q^{1/2}) \sum_{c=0}^{q-1} \exp(2\pi iac/q)|c> \qquad (3.4)$$

Illustrating both the first and second registers, we now have:

$$(1/q) \sum_{a=0}^{q-1} \sum_{c=0}^{q-1} \exp(2\pi iac/q)|c> |x^a \text{ (mod N)}> \qquad (3.5)$$

Step Four: Apply the Inverse Fourier Transform on the first register. Then measure the first register.

$$\phi = c/q = \left| (1/q) \sum_{a=0}^{q-1} \exp(-2\pi iac/q) \right|^2 \qquad (3.6)$$

Using Shor's paper [Sho96], we get the relation that $|c/q - d/r| \leq 1/(2q)$, where r is the order, with $r < N$.

Step Five: We know both the values of c and q. Expand the fraction c/q using the classical continued fraction algorithm. Round c/q to the nearest fraction close to d/r. (Note: This classical algorithm requires polynomial time complexity). We wind up with some d'/r' $\approx$ d/r. If r

is odd, return r. The gcd(r, N) can later be used to determine the factor. Otherwise…

Step Six: When r is even and $x^{r/2} \neq -1 \pmod{N}$, compute:

$$\gcd(x^{r/2} - 1, N) \text{ and } \gcd(x^{r/2} + 1, N). \quad (3.7)$$

If one of these is a non-trivial factor, return the factor. Otherwise, the algorithm fails.

Steps One through Five comprise the Order-Finding Algorithm using the Fourier transform to find the phase estimation.

In the next section, we examine a quantum algorithm used to search for data. The Grover algorithm does not use a Fourier transform technique but, rather, a simplified form of amplitude amplification.

Chapter 4: Grover's Search Algorithm

4.1 Introduction to the Algorithm

Grover's search algorithm [Gro97] uses quantum mechanics to perform a search on data. The algorithm does not use a full Fourier transform like earlier quantum algorithms. Rather, the search algorithm uses a combination of Hadamard transforms and phase rotations to increase or decrease the amplitudes of the quantum states representing the search items. After each iteration of the algorithm, the amplitude of the desired state is increased until it stands apart from the average amplitude of the other search items.

The primary advantage of the algorithm is its relative simplicity of implementation compared to the Fourier transform methods. Given N search items, Grover's algorithm requires O($\sqrt{N}$) time complexity to locate the desired search item, a polynomial time speedup compared to the O(N) time complexity required of classical search algorithms. The simpler Hadamard and phase rotation matrices comprise what the author calls a "Diffusion transform." The Diffusion transform causes the amplitude of the search item to be inverted about the average amplitude of the other search items.

4.2 Steps of the Algorithm

We now list the steps of the Grover Algorithm.

Step One. Initialization. For N search items, we create a superposition of N quantum states, each with $1/\sqrt{N}$ amplitude. For each quantum state, there is also a corresponding location in classical memory with contents describing the properties of the state.

Step Two. Amplitude amplification.

Repeat $\sqrt{N}$ times.

Assume the system is in state |S>.

If a desired property of (S) (from the classical memory contents) matches the search criterion,
ROTATE the phase by π radians;

Otherwise, leave the system unaltered.

Apply the Diffusion Transform matrix, D, to the state as follows:

$D_{ij} = 2/N$ if $i \neq j$ and $D_{ii} = -1 + 2/N$.

Note: The Diffusion matrix can be implemented as a product of three matrices:

D = HRH

…where H is the Hadamard transformation, and R is a rotation matrix.

Step Three. Perform measurement of the resulting state. The measurement will return the desired search item with a probability of at least 50%.

In the next chapter, we will discuss in detail the quantum algorithm for solving a system of linear equations. Unlike the Grover search algorithm, the algorithm of Harrow, Hassidim, and Lloyd uses a full Fourier transform for phase estimation.

Chapter 5: The Quantum Linear Algorithm

5.1 Introduction to the Quantum Linear Algorithm

In their paper, "Quantum Algorithm for Linear Systems of Equations," Harrow, Hassidim, and Lloyd [HHL09a] proposed a quantum algorithm for solving a system of N linear equations with N unknowns. Solving a linear system of equations entails finding a vector, $\vec{x}$, that solves the linear system $A\vec{x} = \vec{b}$, where A is a given matrix operator and $\vec{b}$ is a given vector with constant coefficients.

Quantum states are used to represent $\vec{b}$ and $\vec{x}$. Operator A is sparse, unitary and Hermitian. For example, a 2 x 2 system of linear equations would be represented as:

$$|b\rangle = \sum_{j=1}^{2} b_j |j\rangle \tag{5.1}$$

$$A = \begin{bmatrix} A & 0 \\ 0 & A^{+} \end{bmatrix} \tag{5.2}$$

$$|x\rangle = \text{(unknown states)} \tag{5.3}$$

The sparseness of matrix A is measured by the condition number, κ, defined as the ratio between A's largest and smallest eigenvalues. A small κ indicates a sparse matrix. Using Hadamard gates, a given $|b_j\rangle$ would be rotated into the basis

states of operator A. Let $|u_j\rangle$ be the basis vectors of A, where j = 1…N. Then the states $|b\rangle$ would be represented in the A basis as:

$$|b\rangle = \sum_{j=1}^{N} \beta_j |u_j\rangle \tag{5.4}$$

We define the state, $|\psi_0\rangle$ to be:

$$|\psi_0\rangle = \sqrt{\frac{2}{T}} \sum_{\tau=0}^{T-1} \sin\left[\frac{\pi(\tau + 1/2)}{T}\right] |\tau\rangle \tag{5.5}$$

for a large period T. We apply a conditional Hamiltonian time evolution operator, $\sum_{\tau=0}^{T-1} |\tau\rangle\langle\tau| \otimes \exp(i A \tau t_0/T)$, to the tensor $|\psi_0\rangle \otimes |b\rangle$, with $t_0 = O(\kappa/\varepsilon)$. Using λ_j as the eigenvalue of the operator, and $\beta_j |u_j\rangle$ as the target state, the result of the Hamiltonian for a specific j is:

$$\sqrt{\frac{2}{T}} \sum_{\tau=0}^{T-1} \exp(i \lambda_j \tau t_0/T) \sin\left[\frac{\pi(\tau + 1/2)}{T}\right] |\tau\rangle \beta_j |u_j\rangle \tag{5.6}$$

Phase estimation is then performed using a black box version of the Fourier transform circuit (see the diagram of the quantum Fourier circuit in the next chapter). The multi-qubit phase represents the eigenvalue for a particular solution state (indexed by j). Phases are calculated for each j as follows:

$$|\lambda\rangle\, \beta\, |u\rangle \;=\; \sqrt{\frac{2}{T}} \sum_{j=1}^{N} \;\sum_{\tau=0}^{T-1} \;\sum_{k=0}^{T-1} \; \exp\!\left(i\, \frac{\tau}{T}(\lambda_j t_0 - 2\pi k)\right) \sin\!\left[\frac{\pi(\tau + 1/2)}{T}\right] |k\rangle\, \beta_j\, |u_j\rangle \tag{5.7}$$

Once the phase estimation has been obtained, we set $\lambda_k = 2\,\pi\, k/t_0$ and relabel the $|k\rangle$ basis state as the $|\lambda_k\rangle$ basis state. A qubit is added for each j, and a rotation conditioned on $|\lambda_k\rangle$ is performed. We then undo the phase estimation and uncompute $|\lambda_k\rangle$. Assume we have a perfect phase estimation, such that…

If $\lambda_j = \lambda_k$ then…

$$\sqrt{\frac{2}{T}} \sum_{\tau=0}^{T-1} \; \exp\!\left(i\, \frac{\tau}{T}(\lambda_j t_0 - 2\,\pi\, k)\right) \sin\!\left[\frac{\pi(\tau + 1/2)}{T}\right] = 1 \tag{5.8}$$

…with Equation 5.8 being 0 otherwise.

We measure the last qubit, and obtain λ_j^{-1} (by conditioning on seeing 1). We thus have the form:

$$|x\rangle \;=\; \sum_{j=1}^{N} \; \lambda_j^{-1}\, \beta_j\, |u_j\rangle \tag{5.9}$$

$|x\rangle$ contains the solution to the linear system as a superposition of states in the A-basis vectors. We may then measure one of the properties of $|x\rangle$, obtaining the expectation of that property.

We use the algorithm from Berry et. al [BACS06] to simulate Hamiltonian time evolution. According to the error analysis in [HHL092b], to simulate the evolution of e^{iAt} with an error less than ε, we require that the time t_H be:

$$t_H = O(\log_2(N)\kappa^2 t_0) \tag{5.10}$$

Phase estimation is assumed to be the dominant source of error. When A is sparse, phase estimation can be done with error ε in time proportional to $t\kappa^2(t/\varepsilon)^{O(1)}$. This can be approximated as $O(\kappa^2 t_0)$. Since there are $\log_2(N)$ data registers, the total time required for error ε is $t_H = O(\log_2(N)\kappa^2 t_0)$. That is, the Hamiltonian time evolution must not occur beyond t_H for a desired error, ε.

In the Literature Review we mentioned that log(N) data registers are required, rather than N, since a data register consists of qubits. Since there are log(N) data registers, the algorithm has exponential performance improvement, O(log N), over classical algorithms. The authors point out that exponential speedup over classical algorithms is possible when A is sparse and a single measurement of a property in $|x\rangle$ is performed (rather than multiple measurements). However, if our goal was to obtain all N measurements from $|x\rangle$, the performance would degrade to O(N), on a par with classical algorithms.

The initial preparation of matrix $|b\rangle$ must be performed in a way that is not too time-consuming. If $|b\rangle$ must explicitly reflect classical data, there is no performance advantage over classical methods. However, if one implicitly creates

|b> using amplitudes that are probability distributions, a performance gain over classical methods is possible [GR02].

In addition to the $\log_2(N)$ data registers, a single quantum register implements the unitary operations for all data registers. The state of this unitary operations register does not itself change.

A scratchpad register is also used as a temporary storage location. The scratchpad register is partitioned into log(N) sections, one for each of the logical N quantum variables of the state |x>. Each partition is accessed by the j^{th} data register.

Now that we have provided a general description on some of the features of the algorithm, let us describe the exact steps of the quantum algorithm for linear systems. Figure 4 gives pseudo-code for the algorithm. Section 5.2 provides a detailed description.

Pseudo-Code Description of Quantum Linear Systems Algorithm

Step One. Initialize data registers with $|\psi_0\rangle \otimes |b_j\rangle$.

$$\text{Let } s(\tau) = \sqrt{\frac{2}{T}}\ \sin\left[\frac{\pi(\tau + 1/2)}{T}\right], \tau \text{ held constant.}$$

A given data register contains $s(\tau)\ \beta_j\ |u_j\rangle$.

Step Two. Apply Hamiltonian evolution operator with τ held constant, and apply the Fourier transform with $|k\rangle$ basis states.

Consider separately the cases where $\tau = 0, 1, 2, t_H$.

$$|\lambda\rangle\ s(\tau)\ \beta|u\rangle\ =\ \sum_{j=1}^{N}\ \sum_{k=0}^{T-1}\ \exp\left(i\ \frac{\tau}{T}(\lambda_j t_0 - 2\pi k)\right)\ |k\rangle\ s(\tau)\ \beta_j\ |u_j\rangle$$

Step Three. Apply inverse Fourier transform to obtain $|\lambda_j\rangle$.

Step Four. Relabel $|k\rangle$ with $|\lambda_k\rangle$. Add a qubit for j^{th} variable. Rotate qubit conditioned on $|\lambda_k\rangle$.

Let $\alpha_{k|j} = \exp\left(i\ \frac{\tau}{T}(\lambda_j t_0 - 2\pi k)\right)$. Then:

$$\sum_{j=1}^{N}\ \sum_{k=0}^{T-1}\ \alpha_{k|j}\ |\lambda_k\rangle\ s(\tau)\ \beta_j\ |u_j\rangle\ \left(\sqrt{1 - \frac{C^2}{\lambda_k^2}}\ |0\rangle + \frac{C}{\lambda_k}\ |1\rangle\right).$$

Step Five. Uncompute $|\lambda_k\rangle$ and determine λ^{-1}.

If $\lambda_k = \lambda_j$, then $\alpha_{k|j} = 1$, otherwise $\alpha_{k|j} = 0$. Result is:

$$\sum_{j=1}^{N}\ s(\tau)\ \beta_j\ |u_j\rangle\ \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}}\ |0\rangle + \frac{C}{\lambda_j}\ |1\rangle\right)$$

Measure the last qubit, condition on 1.

$$\lambda^{-1} = \frac{C}{\lambda_j}\ \sqrt{\frac{|\lambda_j|^2}{\sum_{j=1}^{N} C^2\ |B_j|^2\ s(\tau)^2}}\ \Rightarrow |x\rangle\ =\ \sum_{j=1}^{N}\ s(\tau)\ \beta_j\ \lambda^{-1}\ |u_j\rangle$$

Step Six. Measure for the desired property, M: $\langle x|\ M\ |x\rangle$.

Figure 4. Pseudo-Code for Quantum Linear Systems Algorithm

5.2 Detailed Steps of the Algorithm

We assume that the matrix A is already sparse (with κ sufficiently low), and that the states $|b\rangle$ have already been prepared. There are $\log_2(N)$ data registers, representing each unknown variable. We calculate the optimal time for running the Hamiltonian time evolution operator, $t_H = O(\log(N)\ \kappa^2\ t_0)$, and use this time as the upper bound on simulation time.

There is a separate quantum circuit implementing phase estimation for $\tau = 0$, $\tau = 1$, $\tau = 2$, …., $\tau = t_H$. The quantum circuit implementing phase estimation will serve as a "snapshot" of particular times and phases.

Resource issues such as garbage collection, and the classical and quantum variables associated with each step, will be covered in our resource analysis in Chapter 6.

Step One: For each quantum data register, establish states: $|\psi_0\rangle \otimes |b\rangle$.

The qubits representing a particular $|b_j\rangle$ are loaded into the quantum circuit. A Hadamard transformation is performed, decomposing $|b_j\rangle$ and rotating $|b\rangle$ into basis states of operator A. Thus, $|b_j\rangle$ is transformed into $\beta_j\ |u_j\rangle$.

Let $s(\tau) = \sqrt{\frac{2}{T}}\ \sin\left[\frac{\pi(\tau + 1/2)}{T}\right]$, τ held constant.

The unitary operators in the circuitry act on $s(\tau)\ \beta_j\ |u_j\rangle$.

Step Two: The conditional Hamiltonian is applied on $|\psi_0\rangle \otimes |b\rangle$ for constant τ. Also apply the Fourier transform on the conditional Hamiltonian, adding the basis states, $|k\rangle$. The Hamiltonian and Fourier transform are implemented

in the quantum circuit performing phase estimation (see diagrams in Chapter 6).

We have a separate quantum circuit for each time, τ. We consider only the cases where $\tau = 0$, $\tau = 1$, $\tau = 2$, $\tau = t_H$.

In general, the quantum circuit estimates the phase of the eigenvalue, $|\lambda\rangle$, by using 5.11a:

$$|\lambda\rangle\, s(\tau)\, \beta|u\rangle = \sum_{j=1}^{N} \sum_{k=0}^{T-1} \exp\left(i \frac{\tau}{T}(\lambda_j t_0 - 2\pi k)\right) |k\rangle\, s(\tau)\, \beta_j |u_j\rangle \qquad (5.11a)$$

Or, alternately:

$$|\lambda\rangle = \sum_{j=1}^{N} \sum_{k=0}^{T-1} \exp\left(i \frac{\tau}{T}(\lambda_j t_0 - 2\pi k)\right) |k\rangle \qquad (5.11b)$$

For case $\tau = 0$:

$$\sqrt{\frac{2}{T}} \sum_{j=1}^{N} \sum_{k=0}^{T-1} \exp(i\,(0)/T(\lambda_j t_0 - 2\pi k)) \sin\left[\frac{\pi(0+1/2)}{T}\right] |k\rangle\, \beta_j |u_j\rangle$$

$$= \sqrt{\frac{2}{T}} \sum_{j=1}^{N} \sum_{k=0}^{T-1} \sin\left[\frac{\pi(1/2)}{T}\right] |k\rangle\, \beta_j |u_j\rangle$$

Let $s(0)$ be a constant, such that $s(0) = \sqrt{\frac{2}{T}} \sin\left[\frac{\pi(1/2)}{T}\right]$.

Then:

$$|\lambda\rangle\, s(0)\, \beta\, |u\rangle = \sum_{j=1}^{N} \sum_{k=0}^{T-1} |k\rangle\, s(0)\, \beta_j |u_j\rangle. \qquad (5.11c)$$

For case $\tau = 1$:

Let $s(1) = \sqrt{\frac{2}{T}} \sin\left[\frac{\pi(3/2)}{T}\right]$. Then:

$$\sqrt{\frac{2}{T}} \sum_{j=1}^{N} \sum_{k=0}^{T-1} \exp(i\, T^{-1}\, (\lambda_j t_0 - 2\pi k)) \sin\left[\frac{\pi(3/2)}{T}\right] |k\rangle\, \beta_j |u_j\rangle$$

$$= \sum_{j=1}^{N} \sum_{k=0}^{T-1} \exp(i\, T^{-1} (\lambda_j t_0 - 2\pi k)) |k\rangle\ s(1)\, \beta_j |u_j\rangle. \qquad (5.12)$$

For case $\tau = 2$:

Let $s(2) = \sqrt{\frac{2}{T}} \sin\left[\frac{\pi(5/2)}{T}\right]$. Then:

$$\sqrt{\frac{2}{T}} \sum_{j=1}^{N} \sum_{k=0}^{T-1} \exp(i\, \frac{2}{T} (\lambda_j t_0 - 2\pi k)) \sin\left[\frac{\pi(5/2)}{T}\right] |k\rangle\, \beta_j |u_j\rangle$$

$$= \sum_{j=1}^{N} \sum_{k=0}^{T-1} \exp(i\, \frac{2}{T} (\lambda_j t_0 - 2\pi k)) |k\rangle\ s(2)\, \beta_j |u_j\rangle. \qquad (5.13)$$

For case $\tau = t_H$:

Let $s(t_H) = \sqrt{\frac{2}{T}} \sin\left[\frac{\pi(t_H + 1/2)}{T}\right]$. Then:

$$\sqrt{\frac{2}{T}} \sum_{j=1}^{N} \sum_{k=0}^{T-1} \exp(i\, t_H/T(\lambda_j t_0 - 2\pi k)) \sin\left[\frac{\pi(t_H + 1/2)}{T}\right] |k\rangle\, \beta_j |u_j\rangle$$

$$= \sum_{j=1}^{N} \sum_{k=0}^{T-1} \exp(i\, \frac{t_H}{T} (\lambda_j t_0 - 2\pi k)) |k\rangle\ s(t_H)\, \beta_j |u_j\rangle. \qquad (5.14)$$

Step Three: Apply the inverse Fourier transform to obtain the multi-qubit state $|\lambda_j\rangle$, estimating the phase. Refer to the diagram for phase estimation in Chapter 6. To apply the inverse Fourier transform, we start with the multi-qubit result on the right, and read through the diagram backwards, from right to left, through the inverse unitary operators and through the inverse Hadamard transform. The leftmost result is the multi-qubit representation of the phase. The phase for the j^{th} data register is:

$$|\lambda\rangle = |\lambda_0 \lambda_1 \ldots \lambda_{t-1}\rangle$$

When measured, the multi-qubit state gives the phase value.

Step Four: Add a qubit and rotate conditioned on $|\lambda_k\rangle$.

Define $\alpha_{k|j} = \exp(i \frac{\tau}{T}(\lambda_j t_0 - 2\pi k))$.

First we replace $|k\rangle$ with $|\lambda_k\rangle$ by setting $\lambda_k = 2 \pi k / t_0$. We add a qubit associated with the j^{th} variable. Then we rotate conditioned on $|\lambda_k\rangle$, giving us the result:

$$\sum_{j=1}^{N} \sum_{k=0}^{T-1} \alpha_{k|j} |\lambda_k\rangle s(\tau) \beta_j |u_j\rangle \left(\sqrt{1-\frac{C^2}{\lambda_k^2}}|0\rangle + \frac{C}{\lambda_k}|1\rangle\right) \qquad (5.15)$$

Step Five: Undo the phase estimation, $|\lambda_k\rangle$, measure the last qubit, and then determine the inversion factor, λ^{-1}.

Under an ideal phase estimation, $\alpha_{k|j} = 1$ whenever $\lambda_k = \lambda_j$. $\alpha_{k|j} = 0$ otherwise. We assume an ideal phase and drop $|\lambda_k\rangle$. The result is:

$$\sum_{j=1}^{N} s(\tau) \beta_j |u_j\rangle \left(\sqrt{1-\frac{C^2}{\lambda_j^2}}|0\rangle + \frac{C}{\lambda_j}|1\rangle\right) \qquad (5.16)$$

Now we measure the last qubit, and condition on 1. This results in the state:

$$\sqrt{\frac{1}{\sum_{j=1}^{N} C^2 |B_j|^2 s(\tau)^2 / |\lambda_j|^2}} \sum_{j=1}^{N} s(\tau) \beta_j \frac{C}{\lambda_j} |u_j\rangle \qquad (5.17)$$

Equation 5.17 corresponds to $|x\rangle = \sum_{j=1}^{N} s(\tau) \beta_j \lambda^{-1} |u_j\rangle$.

Step Six. We have "solved" for $|x\rangle$ in Step Five, but the values are unknown until measured. We now measure for a single property, M, of $|x\rangle$. Perform $\langle x| M |x\rangle$, obtaining probability(M) for that property.

Example: We can obtain $\langle p_x \rangle$ from $\langle x| p_{op} |x\rangle$.

In the next chapter we will analyze the classical and quantum resources for the algorithm. Classical and quantum variables, garbage collection, and the associated quantum circuits will be considered.

Chapter 6: Analysis of Computational Resources

6.1 Classical vs. Quantum Resources

In this chapter we distinguish the parts of the quantum linear algorithm that are performed classically and the parts that are performed by quantum circuitry. We refer to the quantum objects as quantum numbers or quantum variables. The familiar classical objects are referred to as classical numbers or classical variables.

A classical computer serves as the central control of the implementation, calling the oracular services of quantum circuitry as needed. Quantum "oracles" (represented as the black-box operators in quantum circuitry) are the quantum equivalent of subroutines.

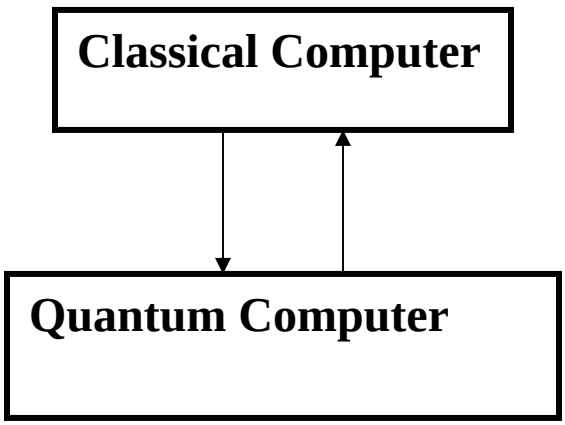

Figure 5: Classical Computer (Client) and Quantum Computer (Server)

Classical computers are used as the control mechanism, rather than quantum computers, due to a relatively fast clock speed. Quantum clock speeds are considerably slower than classical clock speeds, since quantum circuit technology is still in its infancy. The slower quantum clock speeds may well be the case in the near-term future [BCDP96].

In general, the classical computer maps the results of quantum computing (i.e., the measurements of quantum states) into classical memory space, and cross-references the classical and quantum variables. The classical computer also deals with index or control variables that regulate the loop or calling of subroutines. For example, the classical computer may send output to a quantum computer in the form of a program to control the circuitry.

The classical computer may perform the calculations that determine the initial amplitudes of quantum states. (An ongoing design issue is to decide which calculations should be performed classically, and which computations should be incorporated into the quantum circuitry).

To fully analyze the classical and quantum resources, we will track classical and quantum objects at each step of the quantum linear algorithm. We will also indicate when garbage collection becomes necessary during the algorithm, and will discuss the intermediate steps required to eliminate excess quantum states.

The total time to complete a step is the sum of both the classical and the quantum time requirements. The time complexity of each step, and the resulting total time complexity of the entire algorithm, will be estimated.

6.2 Quantum Circuits

Before analyzing the classical and quantum resources of the algorithm, let us first describe the details of a generic quantum Fourier circuit. Figure 6 shows the quantum Fourier transform circuit. This circuit is designed for only one eigenvalue (a single quantum variable, and not a system of equations). The diagram is followed from left (input) to right (output). The box labeled "H' represents the Hadamard transform on the input qubit. The boxes labeled "R_k" represent the unitary transforms of the circuit:

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i / 2^k} \end{bmatrix} \qquad (6.1)$$

The output at the top wire of the diagram represents the least significant qubit, and those wires at the lower portion represent the most significant qubits. Thus, the order of the qubits must be reversed by a swap gate (not shown). Also not shown in Figure 6 are the normalization factors for $\frac{1}{\sqrt{2}}$.

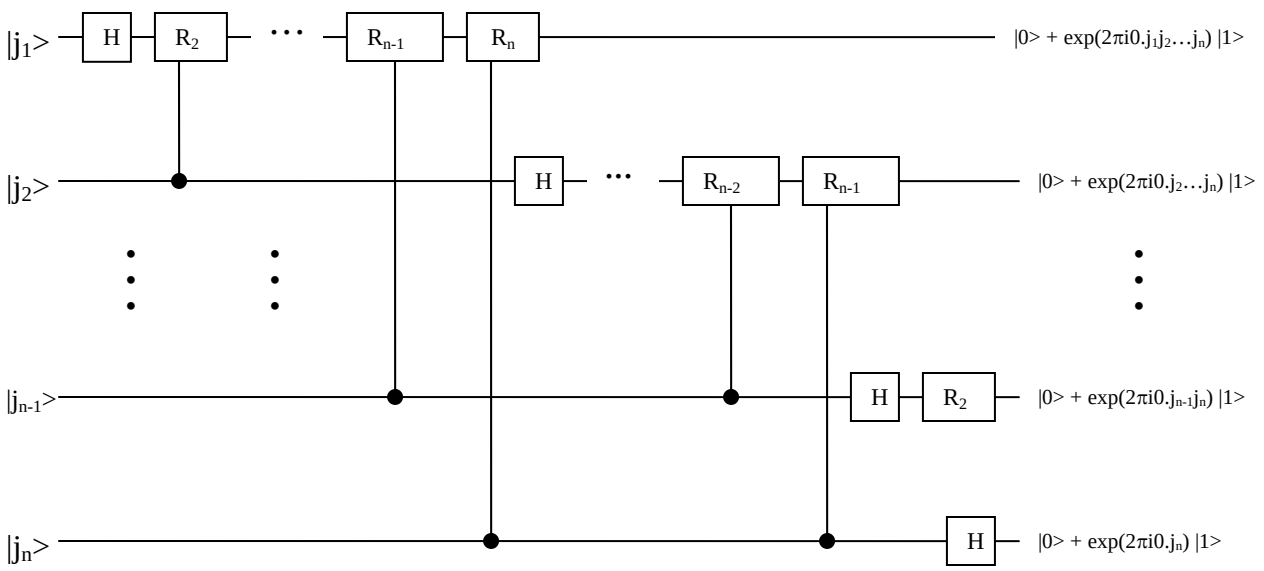

Figure 6: Quantum Fourier Transform Circuit

The quantum Fourier transform of Figure 6 may also be represented by an equivalent product representation and summation representation.

The summation representation can be expressed as:

$$|j_1 \ldots j_n\rangle \rightarrow \sum_{k=0}^{2^{n-1}} e^{2\pi i j k / 2^n} |k\rangle \qquad (6.2)$$

…where $|j_1 \ldots j_n\rangle$ represents the multi-qubit final phase state, and $|k\rangle$ is the Fourier basis state. The equivalent product of qubits representation is:

(6.3)

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

The quantum Fourier transformation can be generalized into a phase estimation. In Figure 7, we replace the R_k operators with black box unitary transformations. As mentioned earlier, a black box is also called an "oracle" (the quantum equivalent of a subroutine). The advantage of representing operations as oracles is that the oracles simplify the representation. Oracles free us from the specifics of implementation, allowing us to focus on the problem at hand.

A major difference between Figure 6 and Figure 7 should be noted. Figure 6 deals with only a single result (a single phase). Figure 7 deals with N phases, since it implements the phase estimations for the N x N linear system of equations.

Again referring to Figure 7, there are $\log_2(N)$ data registers which contain the data (represented as unmeasured quantum states) for all N linear equations and N quantum variables in the system. The index variable, j, refers logically to the j^{th} quantum variable (ranging from 1 to N), although implementation-wise there are log(N) registers.

There is only one register implementing both the unitary transformations and Hamiltonian evolution; the register consists of as many qubits as required to perform the operations. The quantum circuit contains the normalization factors and time constant, $s(\tau)$. The one register performing the unitary transformations is interfaced

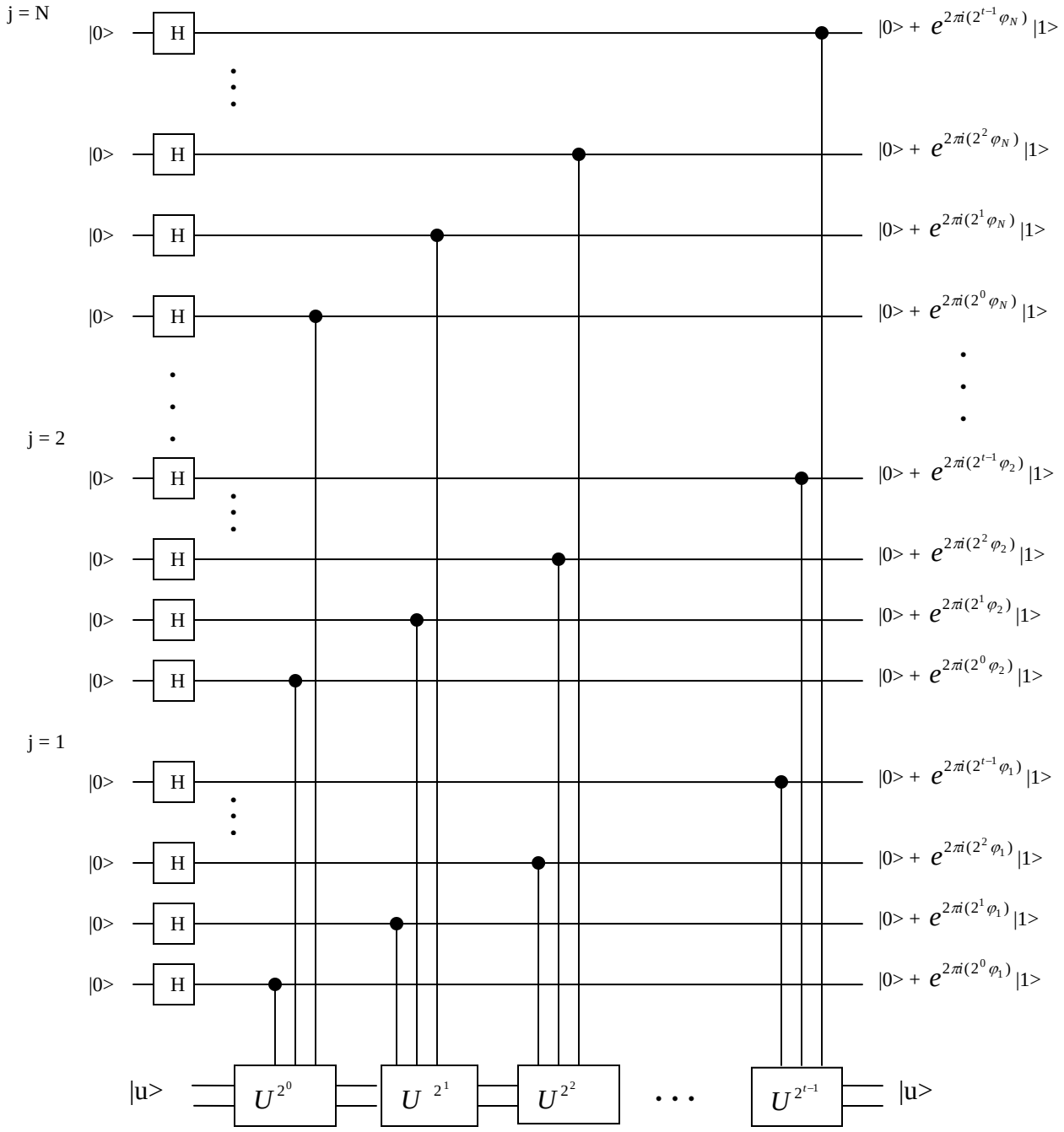

Figure 7. Quantum Phase Estimation for Linear System Algorithm

to all the data registers and can perform the unitary transformations in parallel for the N linear equations, taking advantage of quantum entanglement among the data registers. However, the register that performs unitary operations does not itself change state.

The unitary operator for the phase estimation will have eigenvector |u>, in the A-operator basis, and eigenvalue $e^{2\pi i\phi k}$. ϕ is the phase to be estimated by the quantum circuit. There are N phases to be determined. The unitary operator is implemented by the "black boxes" labeled U^{2^0}, U^{2^1}, U^{2^2}, ..., $U^{2^{t-1}}$. A given phase is estimated by t qubits. The number of qubits required is predicated on how much accuracy we want and the probability for success. The greater the t, of course, the higher the level of accuracy and probability for success.

Although the Hamiltonian evolution operators ideally work with a time sequence, we implement the evolution as a "snapshot" of quantum circuit behavior. The unitary operators are designed to operate with a constant time, τ. I.e., there is a separate quantum circuit for each τ. For illustration purposes, we will consider the behavior of the circuit for $\tau = 0, 1, 2$, and t_H (with t_H being the upper bound on time).

Not shown in Figure 7 are the normalization factor $\frac{1}{2^{t/2}}$ and the time factor $s(\tau)$ (with τ held constant), for the cases $\tau = 0, 1, 2, t_H$.

The summation expression equivalent to Figure 7 for a *particular phase*, j (this time including the normalization factor) is:

$$|\phi_1, \phi_2, \ldots\phi_t\rangle_j \rightarrow \frac{1}{2^{t/2}} \sum_{k=0}^{2^t-1} e^{2\pi i \phi_j k} |k\rangle \tag{6.4}$$

Equation 6.5 gives the equivalent multi-qubit product representation for a particular j:

(6.5)

$$|\phi_1, \phi_2, \ldots\phi_t\rangle_j \rightarrow \frac{(|0\rangle + e^{2\pi i 0.\phi_t}|1\rangle)\ (|0\rangle + e^{2\pi i 0.\phi_{t-1}\phi_t}|1\rangle)\ \ldots\ (|0\rangle + e^{2\pi i 0.\phi_1\phi_2\ldots\phi_t}|1\rangle)}{2^{t/2}}$$

We can express the generic phase, ϕ_j, in terms of the phase for our application, λ_j. Since we are holding τ constant, let $c = \tau/T$. Then, using equation 5.11b, for a particular phase estimation of eigenvalue j:

$$\frac{1}{2^{t/2}} e^{2\pi i \phi_j k} = \frac{1}{2^{t/2}} e^{ic(\lambda_j t_0 - 2\pi k)} \tag{6.6}$$

Solving for ϕ_j by taking the natural log of both sides:

$$\phi_j = c\left(\frac{\lambda_j t_0}{2\pi k} - 1\right) \tag{6.7}$$

Now solving for λ_j using equation 6.7:

$$\lambda_j = \frac{2\pi k}{t_0}\left(\frac{\phi_0}{c} + 1\right) \tag{6.8}$$

Each data register consists of an input buffer to the quantum circuitry, which we label $|\bullet\rangle_{in}$, and an output buffer to store the results of the quantum circuit, which we denote as $|\bullet\rangle_{out}$.

A single scratchpad register is used by all data registers to store the temporary results of calculations, to allow reversibility, and to assist in garbage collection (see Figure 8). The scratchpad is split into log(N) partitions, one for each data register. Each partition is variable in length (of some xt qubits, $x \geq 1$). Logically, however, there are N partitions in the scratchpad, with the j^{th} partition accessible by the j^{th} data register, with $1 \leq j \leq N$. A given partition is in turn broken up into two regions, one called the "Scratch" region and the other called "Copy". The region labeled "Scratch" contains the intermediate or temporary results of calculations. It may contain garbage which must be cleared. The region labeled "Copy" is used to copy the final result of a unitary transformation. When garbage collection occurs, the "Scratch" region is cleared and a permanent result is stored in the "Copy" region for future reference. The output in the Copy region, when used along with the input buffer of a data register, insures reversibility for that data register.

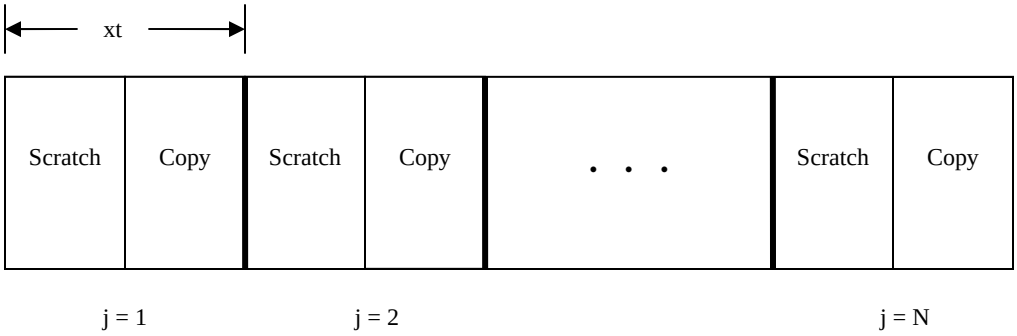

Figure 8. Scratchpad Register

In the following section (6.3) we list the classical and quantum resources required for each step of the linear systems algorithm. The time complexity for the step is also estimated. Upon completion of the algorithm, we determine the order ("Big-O") for the entire algorithm.

We do not estimate space complexity for the algorithm, but note that the total number of qubits used is (t + 1) log(N), excluding the scratchpad, where t is the number of qubits required by each data register.

At the end of this chapter, Table 1 summarizes the constants used in the algorithm. Table 2 summarizes the classical and quantum variables used. Table 3 summarizes the time complexity of each step, and the total time complexity of the entire algorithm.

6.3 Resource Analysis: Tracking Classical and Quantum Objects

Classical Inputs to the Algorithm:

N	Number of rows or columns of linear system; the "size" of the problem. Constant.
T	A sufficiently large time constant; can be assigned to t_H, the maximum value to keep error $\leq \varepsilon$.
τ	A time value held constant for each case of $\tau = 0, 1, 2, t_H$.
t_H	The maximum time value to keep error $\leq \varepsilon$.
t_0	Initial time, a constant.
$s(\tau)$	Constant coefficient; see Step One of Pseudo-Code, Chapter 5.
j	An index referring to data register or row.
β_j	Amplitude of operator A basis states.

Classical Outputs from the Algorithm:

λ^{-1}	Inverse eigenvalue for solution.

Quantum Inputs to the Algorithm:

$\|b_j\rangle$	Basis states of vector $\vec{b}$.
$\|x\rangle$	Unknown states, to be determined and measured.
$\|u_j\rangle$	Basis states of A unitary operator.
$\beta_j \|u_j\rangle$	Vector $\|b\rangle$ in the A-basis.

Intermediate Quantum Variables:

$\|k\rangle$	Fourier basis states
$\|\lambda_j\rangle$	Phase state for eigenvalue of j^{th} data register.

Final Quantum Output:

$\|x\rangle$	Solution states are obtained but are unknown until measured.

Step One. Initialize data registers with $|\psi_0\rangle \otimes |b_j\rangle$.

Classical: As given above in Classical Inputs.

Quantum: As given above in Quantum Inputs.

We first clear all data registers: $|0\rangle_1 |0\rangle_2 \ldots |0\rangle_N$.

After input, a given data register contains $s(\tau)\ \beta_j\ |u_j\rangle$.

Time Complexity:

Let c_0 be the constant time to clear data registers; let c_1 be the time to initialize a data register with $s(\tau)\ \beta_j\ |u_j\rangle$.

We assume that $c_0 << 1 << N$ and $c_1 << 1 << N$. Then:

Time for Step One = $c_0 + c_1 \log(N)$.

Step Two. Apply Hamiltonian with τ constant; apply Fourier transform.

Classical:

t	The maximum number of qubits for each data register j; based on the desired accuracy.
λ_j	Eigenvalue associated with phase for data register j.

Quantum:

$\|k\rangle$	The Fourier transform basis states for data register j.
$\|\lambda_j\rangle$	The multi-qubit phase estimation state.

We consider the cases where $\tau = 0, 1, 2, t_H$. Refer to the detailed explanation of Step Two in Chapter 5, and Figure 7. Assume that there is a separate quantum circuit for each τ. Thus, each circuit operates independently and in parallel to the other circuits. We note that:

Equation 5.11c describes the phase estimation for $\tau = 0$.
Equation 5.12 describes the phase estimation for $\tau = 1$.
Equation 5.13 describes the phase estimation for $\tau = 2$.
Equation 5.14 describes the phase estimation for $\tau = t_H$.

We ignore garbage collection for Step Two but incorporate garbage collection in Step Three. We include the time estimates for garbage collection in Step Three.

Time Complexity:

> Let c_2 be the time for a given data register to complete the Fourier transform, where $c_2 << 1$.
>
> Time for Step Two = $c_2 \log(N)$.

Step Three. Inverse Fourier transform.

Classical: Same as previously described.

Quantum: $|\lambda_{jk}>$, the phase for data register j, for the final k basis state.

$|\bullet>_{in}$, the input buffer for data register j.

$|\bullet>_{out}$, the output buffer for data register j.

$|\bullet>_{scratch}$, the Scratchpad region of Scratchpad j.

$|\bullet>_{copy}$, the Copy region of Scratchpad j.

$f(\lambda_{jk})$, the result of the Fourier transform.

$g(\lambda_{jk})$, the garbage from the Fourier transform.

$f^{-1}(f(\lambda_{jk}))$, or just λ_{jk}, result of the inverse Fourier transform, where f^{-1} is the inverse Fourier transform.

$g^{-1}(f(\lambda_{jk}))$, resulting garbage from inverse Fourier transform, where g^{-1} is the garbage generated by the inverse Fourier transform.

For every data register j, we trace backward through the quantum phase diagram (Figure 7). For each qubit, we start from the right and move to the left, through the inverse unitary operator (which now takes the log), through the inverse Hadamard gate, and out to the left side. Thus, $|\lambda_j>$ is the multi-qubit result of the inverse Fourier transform.

The process of performing the inverse Fourier transform was given as a series of oracular black boxes in Figure 7. The exact details on how the scratchpad was used is kept hidden. However, we assume that the inverse Fourier transform has generated garbage. Let $f(\lambda_{jk})$ be the result of the Fourier transform, and $g(\lambda_{jk})$ be the garbage generated by the Fourier transform. Let $f^{-1}(f(\lambda_{jk}))$, or just λ_{jk}, be the result of the inverse Fourier transform. Let $g^{-1}(f(\lambda_{jk}))$ be the garbage generated by the inverse Fourier transform.

We use the technique of garbage collection described by Beckman et. al [BCDP96]. Let $|\bullet>_{in}$ be the input buffer of data register j. Let $|\bullet>_{out}$ be the output buffer of data register j. Also, $|\bullet>_{scratch}$ and $|\bullet>_{copy}$ are the

scratchpad and copy regions respectively of partition j in the scratchpad register (see Figure 8).

Let F be a unitary operation. Copy is an operation which copies the contents of $|\bullet\rangle_{out}$ to $|\bullet\rangle_{copy}$. The composite operation:

$$F^{-1} \bullet \text{Copy}(|\bullet\rangle_{out}, |\bullet\rangle_{copy}) \bullet F \qquad (6.9)$$

…acts to implement the unitary operation, perform garbage collection, and save both the result and original state (for reversibility).

Let F be the discrete quantum Fourier transform of Step Two.

Initially (prior to operation F), we have the following situation among registers:

$$|\lambda_{jk}\rangle_{in} \; |0\rangle_{out} \; |0\rangle_{scratch} \; |0\rangle_{copy}.$$

After the F is performed:

$$|\lambda_{jk}\rangle_{in} \; |f(\lambda_{jk})\rangle_{out} \; |g(\lambda_{jk})\rangle_{scratch} \; |0\rangle_{copy}.$$

where $g(\lambda_{jk})$ is the garbage generated by $F(\lambda_{jk})$, and $f(\lambda_{jk})$ is the result of the unitary operation.

To perform garbage collection, we first use Copy to duplicate the contents of $|\bullet\rangle_{out}$ to $|\bullet\rangle_{copy}$.

$\text{Copy}(|\bullet\rangle_{out}, |\bullet\rangle_{copy})$ results in the following:

$$|\lambda_{jk}\rangle_{in} \, |f(\lambda_{jk})\rangle_{out} \, |g(\lambda_{jk})\rangle_{scratch} \, |f(\lambda_{jk})_{copy}\rangle.$$

We then use $F^{-1}(\lambda_{jk})$ to clear the output buffer and scratch partition, leaving the input buffer and copy region alone:

$$|\lambda_{jk}\rangle_{in} \, |0\rangle_{out} \, |0\rangle_{scratch} \, |f(\lambda_{jk})\rangle_{copy}.$$

The effect is to get rid of the garbage, $g(\lambda_{jk})$, and to save the useful output to the copy region for future reference.

As mentioned earlier, we assumed that the garbage collection requirements for Step Two were minimal and did not include the time requirements for garbage collection in Step Two.

For Step Three, we incorporate the garbage collection time requirements in our time estimation.

Now let F be the inverse Fourier transform, f^{-1}.

Prior to F being applied to $|f(\lambda_{jk})>$, the following situation exists among the registers:

$$|f(\lambda_{jk})>_{in} \ |0>_{out} \ |0>_{scratch} \ |0>_{copy}.$$

After F is applied:

$$|f(\lambda_{jk})>_{in} \ |\lambda_{jk}>_{out} \ | \ g^{-1}(f(\lambda_{jk}))>_{scratch} \ |0>_{copy}.$$

After Copy is applied:

$$|f(\lambda_{jk})>_{in} \ |\lambda_{jk}>_{out} \ | \ g^{-1}(f(\lambda_{jk}))>_{scratch} \ |\lambda_{jk}>_{copy}.$$

Finally, F^{-1} is applied:

$$|f(\lambda_{jk})>_{in} \ |0>_{out} \ |0>_{scratch} \ |\lambda_{jk}>_{copy}.$$

Time Complexity:

Let c_3 be the time required to perform the inverse Fourier transform on a data register, j.

Let c_g be the time required to perform garbage collection for data register, j, where $c_g \sim O(t^2)$, with t being the length in qubits of the data register j.

Then for a particular data register j, the time required to perform the inverse FT and garbage collection is $c_3 + c_g$. Since there are log(N) data registers:

Total time for Step Three = $(c_3 + c_g) \log(N)$.

Step Four. Relabel $|k>$ with $|\lambda_k>$. Add a qubit for the j^{th} variable. Rotate qubit conditioned on $|\lambda_k>$.

Classical: The address map between classical memory and quantum variables is updated for relabeled name.

New amplitudes for added qubit after rotation:

$\sqrt{1-\frac{C^2}{\lambda_j^2}}$ for state |0>.

$\frac{C}{\lambda_j}$ for state |1>.

λ_j Classical eigenvalue for j^{th} data register.

$C \approx O(1/\kappa)$. A coefficient for probability amplitudes.

Quantum: New qubit established for j^{th} data register.
$|\lambda_k>$ used as pivot for rotation operation.

Relabeling |k> with $|\lambda_k>$ is achieved using the classical computer, which controls the mapping between main memory and corresponding quantum variable. We simply rename the quantum variable, |k>, with $|\lambda_k>$ in the classical address mapping.

Adding a qubit is a quantum operation, repeated physically log(N) times in the quantum circuit. Logically, a qubit is added for each of the N data registers.

Rotation is achieved using a rotation quantum oracle (not shown in Figure 7). This results in a new probability amplitude for the qubit, and also garbage in the scratchpad partition for data register j.

For each data register:

Let $F = Rotate(|\lambda_k>)$. Also, let $F^{-1} = Rotate^{-1}(|\lambda_k>)$. Perform both the rotation and garbage collection using the garbage collection routine described in Step Three as follows:

$$F^{-1}\ Copy(|\bullet>_{out}, |\bullet>_{copy})\ F(|\lambda_k>).$$

Let $\alpha_{k|j} = \exp(i\ \frac{\tau}{T}(\lambda_j t_0 - 2\pi k))$. Then, after the rotation (using Equation 5.15 again):

$$\sum_{j=1}^{N}\ \sum_{k=0}^{T-1}\ \alpha_{k|j}\ |\lambda_k>\ s(\tau)\ \beta_j\ |u_j>\ \left(\sqrt{1-\frac{C^2}{\lambda_k^2}}\ |0> + \frac{C}{\lambda_k}\ |1>\right).$$

The result of the rotation is stored in $|\bullet>_{copy}$, while the original state is kept in $|\bullet>_{in}$ to insure reversibility:

$$|\lambda_k\rangle_{in}\ |0\rangle_{out}\ |0\rangle_{scratch}\ |\text{Rotate}(\lambda_k)\rangle_{copy}.$$

Time Complexity:

Let c_4 be the time required to add a single qubit to j_{th} data register, a quantum operation.

Let c_L be the classic time required to relabel $|k\rangle$ with $|\lambda_k\rangle$.

Let c_r be the time overhead for rotation, including the time required for garbage collection. Time c_r has a classical part (c_r') and a quantum part (c_r''), where $c_r = c_r' + c_r''$. Classical time c_r' involves updating the classical amplitudes and c_r'' involves the quantum circuit overhead.

Total time for Step Four = $(c_4 + c_L + c_r) \log(N)$.

Step Five. Uncompute $|\lambda_k\rangle$ and determine λ^{-1}.

Our resources are described as:

Classical: Measured probabilities of new qubit, conditioned on $|1\rangle$.

λ^{-1} Calculated inverse eigenvalue for solution of j^{th} data register.

Quantum:

$|x\rangle$ Solution set vector to the linear system. Unknown until measured.

Referring to Equation 5.15 again, we assume an ideal phase estimation. When index k = j, then $\alpha_{k|j} = 1$, otherwise $\alpha_{k|j} = 0$.

In terms of classical and quantum resources, we erase those $|\lambda_k\rangle$ states whenever $k \neq j$. We set $|\lambda_k\rangle$ to unity, thus uncomputing $|\lambda_k\rangle$. We leave just the λ_j eigenvalues in the amplitudes for $|0\rangle$ and $|1\rangle$.

After we get rid of $|\lambda_k\rangle$:

$$\sum_{j=1}^{N} s(\tau)\, \beta_j\, |u_j\rangle \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}}\, |0\rangle + \frac{C}{\lambda_j} |1\rangle \right)$$

Then we selectively measure the last qubit of all N data registers. We then condition on state |1> and perform a classical calculation to find λ_j^{-1}. After we condition on state |1> the result is:

$$\sqrt{\frac{1}{\sum_{j=1}^{N} C^2 \mid B_j \mid^2 s(\tau)^2 / \mid \lambda_j \mid^2}} \sum_{j=1}^{N} s(\tau)\ \beta_j \frac{C}{\lambda_j} |u_j>$$

This state corresponds to $|x> = \sum_{j=1}^{N} s(\tau)\ \beta_j\ \lambda^{-1} |u_j>$.

Time Complexity:

Let c_m be the time taken to measure the new qubit.
This time has a classical time (c_m') for calculating probabilities due to measurement, and a quantum time (c_m'') for performing the quantum measurement, where $c_m = c_m' + c_m''$.

Let c_5 be the total time to uncompute $|\lambda_k>$, perform required garbage collection, and calculate λ^{-1}. c_5 has a classical time (c_5') to update the classical address table and to calculate λ^{-1}, and a quantum time (c_5'') to uncompute $|\lambda_k>$ and perform garbage collection. Note that $c_5 = c_5' + c_5''$.

Total time for Step Five = $(c_m + c_5) \log(N)$.

Step Six. Measurement.

We measure for property M, such that <x|M|x> gives us an expectation value for M.

Classical: M, an operator.

Example: M = $\hat{p}_{op}$, such that $<x|\hat{p}_{op}|x> = <p_x>$.

Quantum: Bra-kets for state x.

Time Complexity:

Let c_6 be the time required for quantum measurement and classical table update, where where $c_6 << 1$. c_6 consists of a time for classical table update (c_6') and a time for quantum measurement (c_6''). Note that $c_6 = c_6' + c_6''$.

Total Time for Step Six = $c_6 \log(N)$.

Total Time for Steps One to Six

$$= c_0 + c_1 \log(N) + c_2 \log(N) + (c_3 + c_g) \log(N) + (c_4 + c_L + c_r) \log(N) + (c_m + c_5) \log(N) + c_6 \log(N)$$

$$= c_0 + (c_1 + c_2 + c_3 + c_g + c_4 + c_L + c_r + c_m + c_5 + c_6) \log(N)$$

$$= O(\log(N)).$$

Table 1. Summary of Classical Constants used in Algorithm

Constant	**Description**
T	Large length of time (seconds)
τ	Constant time index in Hamiltonian
t_0	Initial time
$s(\tau)$	Evaluates to constant coefficient:
N	Input size: number of rows or columns
t_H	Maximum time value to keep error $\leq \varepsilon$

Table 2. Summary of Classical and Quantum Resources

Step	**Classical Variables**	**Quantum Variables**	**Comments**
One	j, β_j	A, $\|u\rangle$	j is index to quantum variable; A is Hermitian operator; $\|u\rangle$ represents A-basis vectors; β_j is probability amplitude of $\|u_j\rangle$;
Two	λ_j	$\|u\rangle$, $\|k\rangle$, $\|\lambda_j\rangle$	$\|k\rangle$ are Fourier basis states; $\|\lambda_j\rangle$ is multi-qubit phase estimation. λ_j is eigenvalue for phase.
Three	(same)	In/Out buffers for reg j; Scratchpad	Quantum I/O buffers and Scratchpad used.
Four	C	Added qubit for jth variable; $\|\lambda_k\rangle$	$\|\lambda_k\rangle$ is relabeled Fourier basis state; $C = O(1/\kappa)$, coefficient in amplitude for added qubit, resulting from rotation about $\|\lambda_k\rangle$.
Five	λ^{-1}	$\|\lambda_k\rangle$, $\|x\rangle$	λ^{-1} is inverse phase factor; $\|\lambda_k\rangle$ is uncomputed; $\|x\rangle$ are the solution quantum states;
Six	M	$\|x\rangle$	M is some desired property of data; $\langle x\|M\|x\rangle$ results in expectation of property M.

Table 3. Summary of Time Complexity

Step	**Classical Time**	**Quantum Time**	**Total = Classical + Quantum**
One	N/A	$c_0 + c_1\log(N)$	$c_0 + c_1\log(N)$
Two	N/A	$c_2 \log(N)$	$c_2 \log(N)$
Three	N/A	$(c_3 + c_g) \log(N)$	$(c_3 + c_g) \log(N)$
Four	$(c_L + c_r') \log(N)$	$(c_4 + c_r'') \log(N)$	$(c_4 + c_L + c_r) \log(N)$
Five	$(c_m' + c_5') \log(N)$	$(c_m'' + c_5'') \log(N)$	$(c_m + c_5) \log(N)$
Six	$c_6' \log(N)$	$c_6'' \log(N)$	$c_6 \log(N)$
Total	O(log(N))	O(log(N))	**O(log(N))**

Chapter 7: Summary and Conclusions

This thesis first provided a literature review, covering both the background and recent progress in the field of quantum algorithms. We then focused on a new quantum algorithm that has recently (late 2009) attracted considerable attention in the quantum algorithms community. The quantum algorithm proposed by Harrow, Hassidim, and Lloyd can be categorized as a phase estimation technique (using the Fourier transform as a mechanism for estimating the phase). The algorithm is novel in that it uses the principles of quantum mechanics to solve a system of N x N linear equations. Under optimum conditions (i.e., when we do not desire a readout of all N quantum variables), the algorithm offers exponential time speedup over classical algorithms.

A quantum algorithm that could rapidly solve a system of linear equations has broad applications to a variety of fields.

The contribution of this thesis was to analyze the classical (and related quantum) resources required to implement this quantum algorithm. The analysis clearly distinguished between the tasks required by the classical part of the quantum algorithm, and the quantum resources built into the quantum circuits which implement phase estimation. We determined that the time complexity of the algorithm (including both classical and quantum requirements) for an N x N system of linear equations was O(log(N)), as predicted. Such an analysis may assist future experimentalists in implementing the algorithm.

Furthermore, by analyzing the classical resources, we hope to gain a better understanding on how these resources may be optimized. As classical resources can determine the ultimate efficiency of quantum algorithms, optimizing these resources will assist us in designing more efficient quantum algorithms.

References

[AT03] D. Aharonov and A. Ta-Shma. Adiabatic Quantum State Generation and Statistical Zero Knowledge. arXiv:quant-ph/0301023v2. Submitted 6-7-2003.

[BBCD+95] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin and H. Weinfurter. Elementary Gates for Quantum Computation. *Physical Review A* 52, 3457:3467, 1995.

[BCDP96] D. Beckman, A.N. Chari, S. Devabhaktuni and J. Preskill. Efficient Networks for Quantum Factoring. *Physical Review A* 54, 1034:1063, 1996.

[BBBV97] C. Bennett, E. Bernstein, G. Brassard and Umesh Vazirani. Strengths and Weaknesses of Quantum Computing. *SIAM Journal on Computing 26*, 1510:1523, 1997.

[BV93] E. Bernstein and U. Vazirani. Quantum Complexity Theory. In *25th ACM STOC*, 1993.

[BACS06] D.W. Berry, G. Ahokas, R. Cleve and B.C. Sanders. Efficient Quantum Algorithms for Simulating Sparse Hamiltonians. arXiv:quant-ph/0508139v2. Submitted 2-8-2006.

[BB92a] A. Berthiaume and G. Brassard. The quantum challenge to structural complexity theory. In *Proceedings of the 7th IEEE Conference on Structure in Complexity Theory*, 1992.

[BB92b] A. Berthiaume and G. Brassard. Oracle quantum computing. In *Proceedings of the Physics of Computation*, Dallas, 1992.

[BHT98] G. Brassard, P. Hoyer, and A. Tapp. Quantum Counting. arXiv: quant-ph/9805082v1. Submitted 5-27-1998.

[BHMT00] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp. Quantum Amplitude Amplification and Estimation. arXiv: quant-ph/0005055v1. Submitted 5-15-2000.

[CS96] A.R. Calderbank and P.W. Shor. Good Quantum Error-Correcting Codes Exist. *Physical Review A*, 54:1098, 1996. arXiv:quant-ph/9512032.

[Chi09a] A.M. Childs. Equation Solving by Simulation. *Nature Physics*, Vol. 5, 861, December 2009.

[Chi09b] A.M. Childs. On the Relationship Between Continuous and Discrete-Time Quantum Walk. arXiv:quant-ph/0810.0312v3. Submitted 10-21-2009.

[CCDF+02] A.M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann and D.A. Spielman. Exponential Algorithmic Speedup by Quantum Walk. arXiv:quant-ph/0209131v2. Submitted 10-25-2002.

[CG04] A.M. Childs and J. Goldstone. Spatial Search by Quantum Walk. arXiv: quant-ph/0306054v2. Submitted 8-25-2004.

[CEMM97] R. Cleve, A. Ekert, C. Macchiavello and M. Mosca. Quantum Algorithms Revisited. *Philosophical Transactions of the Royal Society of London*, A (1996). arXiv: quant-ph/9708016v1. Submitted 8-8-1997.

[Deu85] D. Deutsch. Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer. *Proceedings of the Royal Society of London*, Series A, 449, 669:677, 1985.

[Deu89] D. Deutsch. Quantum Computational Networks. *Proceedings of the Royal Society of London*, Series A, 425:473, 1989.

[DJ92] D. Deutsch and R. Jozsa. Rapid solution of problems for quantum computation. *Proceedings of the Royal Society of London*, Series A, 439:553, 1992.

[EJ96] A. Ekert and R. Jozsa. Quantum Computation and Shor's Factoring Algorithm. *Reviews of Modern Physics*, Vol. 68, No. 3, July, 1996.

[FGGS00] E. Farhi, J. Goldstone, S. Gutmann and M. Sipser. Quantum Computation by Adiabatic Evolution. arXiv:quant-ph/0001106. Submitted in 2000.

[Fey82] R. P. Feynman. Simulating Physics with Computers. *International Journal of Theoretical Physics*, Vol. 21, No. 6/7, 1982.

[FT82] E. Fredkin and T. Toffoli. Conservative Logic. *International Journal of Theoretical Physics*, 21, 219:253, 1982.

[GI09] D. Gottesman and S. Irani. The Quantum and Classical Complexity of Translationally Invariant Tiling and Hamiltonian Problems. arXiv:quant-ph/0905.2419v1. Submitted 5-14-2009.

[Gro96] L.K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computation*, ACM Press, New York, 1996.

[Gro97] L.K. Grover. Quantum Mechanics Helps in Searching for a Needle in a Haystack. *Physical Review Letters*, 79(2):325, 1997. arXive e-print quant-ph/9706033.

[GR02] L.K. Grover and T. Rudolph. Creating Superpositions That Correspond to Efficiently Integrable Probability Distributions. arXiv:quant-ph/0208112v1. Submitted 8-15-2002.

[HHL09a] A.W. Harrow, A. Hassidim and S. Lloyd. Quantum Algorithm for Linear Systems of Equations. *Physical Review Letters*, PRL 103, 150502, October, 2009.

[HHL09b] A.W. Harrow, A. Hassidim and S. Lloyd. Supplementary online material for the paper Quantum Algorithm for Linear Systems of Equations.
EPAPS Document No. E-PRLTAO-103-055942.
http://www.aip.org/pubservs/epaps.html.

[Kit95] A.Y. Kitaev. Quantum Measurements and the Abelian Stabilizer Problem. arXive e-print quant-ph/9511026, 1995.

[Llo96] S. Lloyd. Universal Quantum Simulators. *Science*, Vol. 273, 1073 :1078, August, 1996.

[Mer07] N.D. Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, Cambridge, UK, 2007.

[NC00] M.A. Nielsen and I.L. Chuang. Quantum Computation and Quantum Information. Cambridge University Press, Cambridge, U.K., 2000, pp. 120-122.

[Pau03] G. Paun. Membrane Computing. In *Lecture Notes in Computer Science, Fundamentals of Computation Theory*, Vol. 2751, 177:220, Springer, Berlin, 2003.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21 (2), 120:126, 1978.

[Sho94] P.W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. In *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, Santa Fe, NM, 124:134, IEEE Computer Society Press, Nov. 20-22, 1994.

[Sho96] P.W. Shor. Fault Tolerant Quantum Computation. In *Proceedings of the 37th Annual Symposium on Fundamentals of Computer Science*, 56:65, IEEE Press, Los Alamitos, CA, 1996.

[Sim94] D. R. Simon. On the Power of Quantum Computation. In *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, 116:123, 1994.

[Tur36] A. Turing. On Computable Numbers, with An Application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society* 2, 42:230, 1936.

[Yao93] A.C. Yao. Quantum Circuit Complexity. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, 352:361, 1993.