Quantum Computation and a Universal Quantum Computer

Antonio Lagana

School of Chemistry and Physics

The University of Adelaide

A thesis submitted for the degree of

Doctor of Philosophy (Ph.D.)

March 31, 2010

Contents

Glossary

1WQC One-Way Quantum Computing

CPU Central Processing Unit

EPR Einstein Podolsky Rosen

GRK Grover Radhakrishnan

GSQC Ground State Quantum Computation

HQC Holonomic Quantum Computing

I/O Input Output

QAC Quantum Adiabatic Computing

QFT Quantum Fourier Transform

QGA Quantum Gate Array

QTM Quantum Turing Machine

UQC Universal Quantum Computer

UQTM Universal Quantum Turing Machine

UTM Universal Turing Machine

Abstract

This thesis covers two main topics in quantum computing: universal quantum computation and quantum search. We first demonstrate how a quantum harmonic oscillator can be used to implement the universal set of quantum gates and thereby serve as one possible building block for a universal quantum computer. We then address the core and primary focus of this thesis, the theoretical construction of a machine that can compute every computable function, that is, a universal (i.e.*programmable*) quantum computer. We thereby settle the questions that have been raised over the years regarding the validity of the UQTM proposed by Deutsch in 1985. We then demonstrate how to interface the universal quantum computer to external quantum devices by developing programs that implement well-known oracle based algorithms, including the well-known Grover search algorithm, using networked quantum oracle devices. Finally, we develop a partial search oracle and explore symmetry based partial search algorithms utilizing this oracle.

Declaration

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution to myself and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. I give consent to this copy of my thesis when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968. The author acknowledges that copyright of the published work contained in Appendix B of this thesis resides with the copyright holder(s) of the publishing journal. I also give permission for the digital version of my thesis to be made available on the web, via the Universitys digital research repository, the Library catalogue, the Australasian Digital Theses Program (ADTP) and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

The research work presented in this thesis was conducted from February 2007 to March 2010 under the supervision of Associate Professor Max Lohe, Dr. Lorenz von Smekal, and Professor Anthony Williams.

Antonio A. Lagana

March 31, 2010

Adelaide, South Australia, Australia

Acknowledgements

I would like to thank my supervisors, Associate Professor Max Lohe, Dr. Lorenz von Smekal, and Professor Anthony Williams, for their support, guidance, and constructive criticism of my work. I am grateful that they were always available and willing to help whenever I sought it. I would also like to thank Dr. Rod Crewther for supporting my pursuit of a Ph.D. in theoretical physics. I would also like to thank Dr. Rod Crewther and Dr. Lorenz von Smekal for their excellent courses in quantum mechanics, both of which helped me to acquire the required knowledge and understanding of quantum mechanics to conduct my research. Last but not least, I would like to thank the School of Chemistry and Physics for providing the opportunity and facilities to pursue this work.

1

Introduction

1.1 Theoretical Foundation of Quantum Computation

Quantum computing has been and continues to be an active research area [1]. As best exemplified by Shor's factorization algorithm [2], quantum computing algorithms have the potential to achieve significant speed-ups over their classical counterparts. In fact, some have gone as far as suggesting that quantum computability can potentially surpass classical computability by solving problems such as the famous halting problem [3].

It is not yet clearly understood what quantum properties give rise to speed-ups in quantum algorithms. At first glance, it would appear that the quantum properties of superposition and entanglement are responsible and they may very well be. However, superposition can also be realized in classical optical systems (see [4] for an example of a classical optical implementation of Grover's search algorithm [5] using lasers) and entanglement is far from being well understood. In fact, the theory of entanglement is an active research area in and of itself. In any case, whether entanglement plays a crucial role in quantum computing is still not clear (see [6] and [7] or [8] for example).

The popular quantum gate array (QGA) model [9] has been shown to be universal in the sense that any function can be implemented using a set of single-qubit gates (X, Y, Z, and the T or $\frac{\pi}{8}$ gate) and the two-qubit CNOT gate much as the two-bit NAND gate is universal for classical Boolean circuits (see [1] for example). This is a crucial result because it means that quantum computers can compute any function that can be computed by classical computers and in some cases with a significant speed-up. In

this thesis, we explore this gate level aspect of universality by demonstrating how a quantum harmonic oscillator can be used to implement the universal set of quantum gates. A quantum harmonic oscillator can serve as the building block to implement a universal quantum computer like the one that we construct in this thesis to resolve the long-standing "halting problem" of quantum computers.

Perhaps due to the nascent nature of the field, there are many models or frameworks of quantum computing and more are still being introduced. In addition to the popular QGA model, other proposed models of quantum computation include the Quantum Adiabatic Computing (QAC) [10], Ground State Quantum Computation (GSQC) [11], Holonomic Quantum Computation (HQC) [12], Measurement-Based Quantum Computation [13], and Quantum Walks [14] just to name a few.

The basic idea behind QAC is as follows. One first defines an initial Hamiltonian, call it H_0, with a well defined ground state. A "problem" Hamiltonian, call it H_P, is then defined whose ground state (that is both unique and guaranteed to exist) corresponds to the computational problem that is to be solved. Stated differently, H_P encodes the problem to be solved. The Hamiltonian is then evolved adiabatically from H_0 to H_P. By the quantum adiabatic theorem, if the system starts out in the ground state of H_0, the evolution is slow enough, and the energy levels are finitely spaced (i.e. there are non-zero energy gaps between energy levels) throughout the evolution, then at the end of the evolution, the system will be very close to the ground state of H_P. In other words, the system will be in the "solution" state for the problem to be solved. A measurement of the final state then provides the solution to the problem at hand.

The QAC model has been shown to be computationally equivalent to the QGA model [15] [16]. That is, a QAC algorithm can emulate a QGA algorithm in polynomial time. As such, it can be said to be a universal model of quantum computing. Moreover, it has been highlighted that QAC can be speeded up by tuning the evolution path and by increasing the energy of the system [17]. It should be noted that because quantum computation is necessarily reversible (due to the unitarity of quantum evolution), in principle, it involves no energy dissipation or power consumption. Hence, the increased amount of energy to speed up the evolution remains in the system and can, in principle, be recovered after the computation has been completed. Of course, in practice, some energy will be lost in various physical processes associated with running the quantum

computer but the point is that the computation itself does not consume the increased energy. The increased energy simply enables a speed up of the evolution.

Classical computing science complexity analysis considers space (e.g. circuit or memory size) and time (e.g. number of gate stages or operations) resource usage to characterize algorithmic complexity. QGA algorithms are generally characterized and analyzed in terms of the number of quantum gates stages as a function of the problem size. For example, the Grover search algorithm is characterized by the number of oracle calls. QAC algorithms are usually characterized by the evolution time as a function of the problem size. However, since quantum computation can be arbitrarily speeded up by using energy, it needs to be treated on the same footing that space and time resources are treated in classical computing complexity analysis. The role that energy plays in speeding up algorithms appears to have been noticed relatively recently (see [17] for example) but has been restricted to the QAC framework. Yet, energy can speed up QGA algorithms and, in fact, any quantum computer based on quantum evolution.

Not all proposed models for quantum computation rely on time evolution of quits, however. GSQC is such an example. The basic idea behind GSQC is to produce a ground state that spatially encodes the entire temporal trajectory of the algorithm and the steps in the computation correspond to the development of the ground state between parts of the Hilbert space instead of between time points. This model requires a larger number of quits and thus exchanges space complexity in exchange for increased robustness against decoherence.

The HQC model is based on Berry's discovery [18] that the wave function of a quantum system retains the memory of its evolution in its complex phase. Apart from the contribution due to the dynamical evolution, this complex phase only depends on the geometry of the traversed path in Hilbert space. This *geometric phase* is resilient to certain types of errors and thus the idea is to exploit this inherent robustness as a means of constructing fault-tolerant quantum computing components. In the HQC model, the Hamiltonian of the system is varied adiabatically in parameter space to generate arbitrary unitary operations. What is unique about this evolution is that the computational subspace remains degenerate throughout the computation.

Measurement-Based Quantum Computation models were devised to utilize unique features of quantum mechanics, namely entanglement and teleportation. The basic idea behind these models is to use a special entangled state to realize the controlled-NOT

gate via teleportation. One-way quantum computation (1WQC) [19], for example, is based on these ideas. The computation begins by preparation of a special entangled state called a cluster state. The cluster state is essentially a matrix of gates that acts as the computation gates and controls the information flow. The computation steps proceed by making single quit measurements. This model is considered to be simpler to realize in practice. 1WQC has also been shown to be universal [20].

As mentioned above, all these models of quantum computing have been shown to be "universal" in the sense that a machine can be constructed to compute any given computable function but this notion is different from the classical computer science notion of universality. In computer science, universality refers to the fact that every computable function can be computed by a Universal Turing Machine (UTM). According to the Church-Turing Thesis, all finitely realized computing machines can be simulated by a *single* machine, the UTM. Modern computers, while highly sophisticated, are fundamentally UTM implementations. Thus, universality is extremely important because programmability follows from universality.

The QGA computing model is not universal in this sense because it is not programmable. A single QGA computer cannot simulate every other quantum computer. Each QGA computer must be purpose built or configured to implement a particular algorithm. Even QAC is not strictly programmable in this sense because the time dependent Hamiltonian must be individually tailored for each given problem. In this sense, currently envisioned quantum computers more closely resemble FPGAs (Field Programmable Gate Arrays) rather than CPUs (Central Processing Units), to draw an analogy with classical computers.

In one of the founding papers of quantum computation [21], Deutsch defined a Quantum Turing Machine (QTM) and further claimed that there exists a Universal Quantum Turing Machine (UQTM). These were both quantum generalizations of their classical counterparts. The UQTM was defined to be a QTM for which there exists a program as part of its input state that has the effect of applying a unitary transformation on an arbitrary number of qubits arbitrarily close to any desired transformation. That is, the UQTM could simulate, up to arbitrary accuracy, the operation of any given QTM. As its classical counterpart, a QTM contains a halt qubit that is used to indicate whether the computation has been completed. Thus, the UQTM is truly uni-

versal in the computer scientific sense in that it is programmable and able to simulate the operation of every QTM.

Unfortunately, since its original proposal, several questions were raised as to the validity of the UQTM. In 1997, Myers suggested that the UQTM's halting scheme was invalid [22]. He argued that an entanglement between the halt qubit and other qubits could occur, thereby making it impossible to determine whether the machine has halted or not. Furthermore, in 2001 Shi pointed out that the UQTM halting scheme was a special case of the program *concatenation* scheme that was assumed to be valid in the original UQTM proposal. But the validity of the program concatenation scheme itself had not been proven. Shi did not prove or disprove the validity of the program concatenation scheme and this remained an open question. In short, the fundamental question of whether a universal (programmable) quantum computer exists was still an open question until we constructed the Universal Quantum Computer ($\mathcal{UQC}$) presented in this thesis and published in [23].

It is also still unclear whether quantum computability can surpass classical computability in the sense of being able to compute functions that are not computable by classical computers (e.g. the famous Turing halting problem). There have been claims [24] [25] about quantum computability surpassing classical computability (the so-called *hypercomputing*) and counterclaims as to its validity (see [3] for a comprehensive summary of a debate along these lines) but this remains an open question because there is as of yet insufficient theoretical foundation to draw any conclusions. At the very least, it can be said that we need a universal model of quantum computing to draw such conclusions, just as a model of universal computation (i.e. the UTM) enabled formal analysis and investigation of classical computability and complexity. The $\mathcal{UQC}$ may help shed some light on this question and we briefly explore this question by attempting to determine the physical basis for the halting problem.

Following the construction of a standalone universal quantum computer, the next natural next step is to consider the networking of such machines. Quantum networks that connect quantum systems and can transmit quantum information have been extensively discussed [26]. Quantum connectivity provides a means of overcoming size-scaling and error-correction problems, and has significant advantages over classical connectivity. Furthermore, networks of quantum computers have also been proposed [27] where information can be exchanged between nodes via quantum and classical channels. A

general question arises as to how such quantum computers can communicate and exchange information. In the simplest case a quantum computer may download data sets from other nodes over the quantum network, but in more complex cases it might use the network to call subroutines, or concatenate programs from other quantum computers.

We address this question of how a universal quantum computer can access an external oracle, which may be regarded as a "black box" quantum device, possibly over a quantum network but in any case as a separate and external quantum system to the universal quantum computer itself. In fact, the oracle may be a program running on a remote universal quantum computer. We demonstrate how to interface external/remote quantum devices with the $\mathcal{UQC}$ by implementing well known oracle based algorithms, namely the Deutsch, Deutsch-Jozsa, and the Grover search algorithms, using black-box quantum oracle devices that are external/remote to the $\mathcal{UQC}$. We thereby show how the $\mathcal{UQC}$ can utilize networked quantum information resources to carry out local computations. This work was published in [28].

Fast search algorithms are extremely important and thus it is no surprise that the two best known quantum algorithms are Grover search and Shor's factorization algorithms. These two algorithms are significantly faster than all currently known classical counterparts. Unlike Shor's algorithm, Grover search and its variants can at best accomplish a quadratic speedup over the fastest known classical search algorithms. There are well established lower bounds that show that the Grover search algorithm and variants thereof are optimal. However, these bounds are based on the premises that the algorithm consists of a sequence of unitary operations of a particular form and using a particular form of an oracle. As such, these lower bounds do not necessarily apply if non-unitary operations such as projections are introduced or if different oracles are utilized. We thus explore a partial search scheme in the hope of escaping the Grover bound [29]. We develop a partial search oracle based on the standard Grover search oracle and invoking symmetry projections and explore two partial search algorithms based on it. We further find that if non-linear quantum processes are ever observed, and if non-linearity were to provide a way to copy quantum states and deterministically superimpose them in a particular manner, it would be possible to realize exponentially fast search algorithms that surpass all currently known search algorithms.

1.2 Organization of this Thesis

This thesis consists of the following chapters.

1. Introduction

2. Quantum Computation Using The Harmonic Oscillator
   Quantum computation is typically based on implementing qubits using spin-1/2 particles because they lend themselves naturally to encoding binary states. We demonstrate a way to achieve universal quantum computation based on the energy eigenstates of the quantum harmonic oscillator by implementing a universal set of quantum gates using the quantum harmonic oscillator. The motivation for this chapter is to present a possible building block for the construction of a universal quantum computer that is covered in the next chapter.

3. Universal Quantum Computer
   This chapter comprises the core of this thesis and describes the construction of the $\mathcal{UQC}$ following Deutsch's original proposal of a Universal Quantum Turing Machine. We begin by presenting a summary of the original UQTM and the problems raised against it, define the architecture of the $\mathcal{UQC}$, define its instruction set, define the evolution operators that govern the operation of the machine, define some primitive programs that can be used as building blocks for the creation of more sophisticated programs, discuss the program concatenation scheme. Finally, we explore the Church-Turing Thesis and the halting problem in the context of the $\mathcal{UQC}$.

4. Oracle Based Algorithms On A Universal Quantum Computer
   We demonstrate how to interface external/remote quantum devices with the $\mathcal{UQC}$ by implementing well known oracle based algorithms, namely the Deutsch, Deutsch-Jozsa, and the Grover search algorithms, using black-box quantum oracle devices that are external/remote to the $\mathcal{UQC}$. We thereby show how the $\mathcal{UQC}$ can utilize networked quantum information resources to carry out local computations.

5. Symmetry Based Partial Search
   We explore the possibility to escape the lower bound on the number of required oracle calls to perform a partial search of an unstructured database by allowing

for the use of projections, thereby using a non-unitary scheme. We first develop a partial search oracle based on symmetry projections. We then explore two partial search algorithms based on this partial search oracle.

2

Quantum Computation Using The Harmonic Oscillator

2.1 Introduction

Quantum computation is typically based on implementing qubits using spin-$\frac{1}{2}$ particles because they lend themselves naturally to encoding binary states. That is, the unit of information used for computation is the qubit with orthonormal basis states $|0\rangle \stackrel{\text{def}}{=} |\uparrow\rangle$ and $|1\rangle \stackrel{\text{def}}{=} |\downarrow\rangle$. This is a natural extension of classical Boolean circuits so it seems natural that most of the research work in quantum computing would be based on this scheme. The harmonic oscillator is very well understood and it has been widely applied in many areas of quantum physics. Yet attempts to harness the harmonic oscillator for quantum computation purposes are relatively recent (see [30] for example). The motivation for this chapter is to establish the quantum harmonic oscillator as a potential building block for constructing the universal quantum computer in the next chapter.

Universal computation (in the sense of being able to implement any quantum algorithm, not in the programmable sense) can be implemented based on the energy eigenstates of the quantum harmonic oscillator. Beyond its academic interest, harmonic oscillators possess several properties that could have advantages over their spin-$\frac{1}{2}$ counterparts. Firstly, the quantum harmonic oscillator is well understood and is relatively easy to work with from the standpoint of theoretical analysis. Secondly, the energy eigenstates of a harmonic oscillator are equally spaced and non-overlapping which could be harnessed to achieve a level of tolerance to external perturbations or

interactions with its external environment. Finally, the harmonic oscillator eigenstates span an infinite Hilbert space. We can conjecture about the possibility to harness the countably infinite states to surpass finite state computing. As discussed in [31], the resources required to achieve hypercomputation essentially boil down to the use of an infinite number (distinct from arbitrarily large number) of resources such as memory, processor states, computational steps, etc. and being able to use these in a finite amount of time. As a concrete example, the energy eigenstates of the quantum harmonic oscillator encode all the natural numbers (i.e. $\hat{N}|n\rangle = n|n\rangle$). Given any function f of the natural numbers and a natural number n, if $f(\hat{N})$ were observable, then this would provide a way to compute $f(n)$ in constant time. In other words, the quantum harmonic oscillator could be used as a glorified lookup table. This suggests that there are no physical observables that correspond to such functions.

In order to realize universal computation using a harmonic oscillator, we need to devise a way to implement a qubit and to construct a universal gate set that will allow us to implement any algorithm developed for the QGA framework.

2.2 Implementing The Qubit

We first define a qubit using the first two energy eigenstates of the quantum harmonic oscillator. That is, we define $|0\rangle \stackrel{\text{def}}{=} |n = 0\rangle$ and $|1\rangle \stackrel{\text{def}}{=} |n = 1\rangle$ where n denotes the energy level of the harmonic oscillator:

$$E_n = \left(n + \frac{1}{2}\right)\hbar\omega,$$

where E_n is the energy of the n-th eigenstate with $\hbar$ denoting Planck's constant and ω denoting the frequency. The energy eigenstates also form a *number states basis* where

$$\hat{N}|n\rangle = n|n\rangle, \quad n = 0, 1, 2, ...$$

$\hat{N}$ is the *number* operator and $\hat{N} = aa^\dagger$ where a and $a^\dagger$ are the energy lowering and raising operators, respectively.

2.3 Constructing a Universal Gate Set

In order to construct a universal gate set, it suffices to construct the so-called Pauli gates, the Hadamard gate, the T or $\frac{\pi}{8}$ gate, and the CNOT gate [1]. We first construct

the Pauli gates. Recall that the Pauli gates must act on a qubit as follows. The "σ_x" or X gate needs to act as

$$\begin{aligned} \mathrm{X}|0\rangle &= |1\rangle, \\ \mathrm{X}|1\rangle &= |0\rangle. \end{aligned}$$

The "σ_y" or Y gate needs to act as

$$\begin{aligned} \mathrm{Y}|0\rangle &= i|1\rangle, \\ \mathrm{Y}|1\rangle &= -i|0\rangle. \end{aligned}$$

The "σ_z" or Z gate needs to act as

$$\begin{aligned} \mathrm{Z}|0\rangle &= |0\rangle, \\ \mathrm{Z}|1\rangle &= -|1\rangle. \end{aligned}$$

Let $\mathrm{X} = |0\rangle\langle 1| + |1\rangle\langle 0| + \sum_{n=2}^{\infty} |n\rangle\langle n|$. It can be readily verified that this gate works as required and is Hermitian. It is also unitary as shown below.

$$\mathrm{X}\mathrm{X}^\dagger|n\rangle = \begin{cases} \mathrm{X}|(n+1) \mod 2\rangle = |(n+2) \mod 2\rangle = |n\rangle, n \leq 1, \\ \mathrm{X}|n\rangle = |n\rangle, n \geq 2. \end{cases}$$

Let $\mathrm{Y} = -i|0\rangle\langle 1| + i|1\rangle\langle 0| + \sum_{n=2}^{\infty} |n\rangle\langle n|$. It can be readily verified that this gate also works as required and is Hermitian. It is also unitary as shown below.

$$\begin{aligned} \mathrm{Y}\mathrm{Y}^\dagger|n\rangle &= \begin{cases} \mathrm{Y}(-1)^n i|(n+1) \mod 2\rangle, n \leq 1, \\ \mathrm{Y}|n\rangle, n \geq 2, \end{cases} \\ &= \begin{cases} |(n+2) \mod 2\rangle = |n\rangle, n \leq 1, \\ |n\rangle, n \geq 2. \end{cases} \end{aligned}$$

Let $\mathrm{Z} = |0\rangle\langle 0| - |1\rangle\langle 1| + \sum_{n=2}^{\infty} |n\rangle\langle n|$. It can be readily verified that this gate also works as required and is Hermitian. It is also unitary as shown below.

$$\begin{aligned} \mathrm{Z}\mathrm{Z}^\dagger|n\rangle &= \begin{cases} \mathrm{Z}(-1)^n|(n+1) \mod 2\rangle, n \leq 1, \\ \mathrm{Z}|n\rangle, n \geq 2, \end{cases} \\ &= \begin{cases} |(n+2) \mod 2\rangle = |n\rangle, n \leq 1, \\ |n\rangle, n \geq 2. \end{cases} \end{aligned}$$

The Hadamard gate needs to act as

$$\begin{aligned} \mathrm{H}|0\rangle &= \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \\ \mathrm{H}|1\rangle &= \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \end{aligned}$$

It is readily verified that $\mathrm{H} = \frac{1}{\sqrt{2}}(\mathrm{X} + \mathrm{Z})$. Since both X and Z are Hermitian and unitary so is H.

The T gate needs to act as

$$\begin{aligned} \mathrm{T}|0\rangle &= |0\rangle, \\ \mathrm{T}|1\rangle &= e^{i\frac{\pi}{4}}|1\rangle. \end{aligned}$$

Therefore, let

$$\mathrm{T} = |0\rangle\langle 0| + e^{i\frac{\pi}{4}}|1\rangle\langle 1| + \sum_{n=2}^{\infty} |n\rangle\langle n|.$$

It can be readily verified that this gate works as required. Naturally, since this gate only affects the phase, it is not observable and hence is not Hermitian. It is, however, unitary.

$$\mathrm{TT}^{\dagger}|n\rangle = \begin{cases} \mathrm{T}e^{-i\frac{\pi}{4}n}|n\rangle = |n\rangle, n \leq 1, \\ \mathrm{T}|n\rangle = |n\rangle, n \geq 2. \end{cases}$$

Finally, we need a CNOT gate that acts as follows.

$$\mathrm{CNOT}|c\rangle|d\rangle = |c\rangle|d \oplus c\rangle,$$

where the c qubit is considered the *control* and d the *data* qubit. Let

$$\mathrm{CNOT} = (|0\rangle\langle 0|)_c \otimes (|0\rangle\langle 0| + |1\rangle\langle 1|)_d + (|1\rangle\langle 1|)_c \otimes (|1\rangle\langle 0| + |0\rangle\langle 1|)_d + \sum_{m,n=2}^{\infty} (|m\rangle\langle m|)_c \otimes (|n\rangle\langle n|)_t$$

It can be readily verified that this gate works as required and that it is Hermitian. It is also unitary.

$$\begin{aligned} \mathrm{CNOT} \cdot \mathrm{CNOT}^{\dagger}|c\rangle|d\rangle &= \begin{cases} \mathrm{CNOT}|c\rangle|d \oplus c\rangle, c, d \leq 1, \\ \mathrm{CNOT}|c\rangle|d\rangle, c, d \geq 2. \end{cases} \\ &= \begin{cases} |c\rangle|d \oplus c \oplus c\rangle = |c\rangle|d\rangle, c, d \leq 1, \\ |c\rangle|d\rangle, c, d \geq 2. \end{cases} \end{aligned}$$

Thus, it is possible to define a set of gates that are universal and hence harmonic oscillator based quantum computation is universal. All QGA algorithms (e.g. Grover search) can be directly implemented without any modifications because those algorithms can be implemented using the universal gate set.

In fact, it has been shown that a higher-dimensional version of a qubit, the so-called *qudit*, can be encoded using the harmonic oscillator [30]. This makes it possible to exploit the entire accessible Hilbert space. This could have practical advantages in realizing workable quantum computers because it provides for a more efficient encoding than qubit based computing which should result in fewer interacting components in a physically realized quantum computer.

2.4 Discussion

We have shown that the harmonic oscillator can be used to implement a universal set of quantum gates. This can be used as the basis for a physical construction of the $\mathcal{UQC}$ that will be constructed in the next chapter. Moreover, as mentioned earlier, the harmonic oscillator energy eigenstates span an infinite dimensional Hilbert space unlike spin-$\frac{1}{2}$ systems. It is conceivable that the infinite dimensional Hilbert space could result in some useful properties that are absent in spin-$\frac{1}{2}$ systems. However, this is beyond the scope of this thesis and rather than pursuing yet another model of quantum computation, we will now turn our attention to the primary focus of this thesis, the construction of the $\mathcal{UQC}$.

3

Universal Quantum Computer

In one of the founding papers of quantum computation, Deutsch [21] defined a Quantum Turing Machine (QTM) and further claimed that there exists a Universal Quantum Turing Machine (UQTM) that is a natural quantum generalization of the classical UTM, both of which are quantum generalizations of their classical counterparts. The UQTM was defined to be a QTM for which there exists a program as part of its input state that has the effect of applying a unitary transformation on an arbitrary number of qubits arbitrarily close to any desired transformation. That is, the UQTM could simulate, up to arbitrary accuracy, the operation of any given QTM. As its classical counterpart, a QTM contains a halt qubit that is used to indicate whether the computation has been completed. Thus, the UQTM is universal in the sense in that it is programmable and able to simulate the operation of every possible QTM. The theoretical existence of such a machine is important because it would establish whether a programmable quantum computer can be constructed in principle.

Since the original proposal, several questions have been raised as to whether the UQTM, as defined by Deutsch, was indeed valid. In 1997, Myers [22] argued that the UQTM's halting scheme was invalid. In 2001 Shi [32] showed that the validity of the halting scheme ultimately rested on whether the program concatenation scheme was valid. If the concatenation scheme were valid, the halting scheme would be valid, and Myer's question could be resolved by Ozawa's [33] non-demolition measurements of the halting qubit subject to the requirement that the halting scheme be implemented in a way whereby the state of the memory tape ceases to change once the halt qubit is

set. The question of whether the concatenation scheme is valid and hence whether a UQTM exists, has remained an open question.

In the following sections, we first review the QTM and UQTM as defined by Deutsch and then present an explicit construction of a universal quantum computer to demonstrate that a universal (programmable) quantum computer exists and that program concatenation is valid. The machine supports programmatic execution basic instructions that include the universal set of unitary operations as well as a conditional branch instruction. Like Deutsch's UQTM, our machine consists of a memory tape and processor. The internal architecture of our machine is very similar to that of a classic microcontroller and contains a data address register, program counter, status flag, instruction fetch buffer register, and a memory read/write head. In addition, our machine contains a halt qubit that signifies whether program execution has completed and a flow control register and history buffer address register that are used to store program execution history information. The flow control and history buffer address registers are used to store a sufficient amount of information such that, in principle, the operation of any program can be reversed at any given time, consistent with unitarity. This theoretical construction will be useful to analyze other aspects of quantum computation, such as complexity analysis of algorithms, analysis of the halting problem (in the Church-Turing Thesis sense), etc. in an analogous way that a UTM is used in classical computer science.

Sections 3.1 and 3.2 provide brief descriptions of Deutsch's QTM and UQTM, respectively. In 3.3 we describe several problems that were raised about the QTM halting scheme and the fact that it relies on the validity of program concatenation, something that Deutsch did not prove. In 3.4 we present an explicit construction of a universal quantum computer and describe the internal architecture, instruction set, and the time evolution operator associated with the machine. In 3.5 we define a set of basic programs in order to demonstrate the classical universal nature of our machine by constructing a program that computes the NAND function. In 3.6 we discuss program concatenation, and present a program concatenation operator for our machine thus demonstrating that program concatenation is valid for quantum computers.

3.1 The Quantum Turing Machine

As defined by Deutsch, a QTM consists of two components: a finite processor and an infinite tape (external memory), of which only a finite portion is ever used. The finite processor consists of N qubits and the infinite tape consists of an infinite sequence of qubits, of which only a finite portion is ever used. The currently scanned tape location is specified by x which denotes the 'address' number of the tape. Thus, the state of a QTM is a unit vector in the Hilbert space spanned by the basis states $|x\rangle|\mathbf{n}\rangle|\mathbf{m}\rangle$, where $|\mathbf{n}\rangle \stackrel{\text{def}}{=} |n_0, n_1, n_2, ..., n_{N-1}\rangle$, and $|\mathbf{m}\rangle \stackrel{\text{def}}{=} |..., m_{-2}, m_{-1}, m_0, m_1, m_2...\rangle$.

The operation or dynamics of the machine is defined by a *fixed* unitary operator U whose only non-trivial matrix elements are $\langle x \pm 1; \mathbf{n}'; m'_x, m_{y\neq x}|U|x; \mathbf{n}; m_x, m_{y\neq x}\rangle$. That is, only one tape qubit, the x-th, participates in any given computational step and at each step, the position of the head cannot change by more than one unit, forward or backward, or both in the case that the position of the tape is a superposition of $|x \pm 1\rangle$. Each different U corresponds to a *different* QTM. Stated differently, each QTM corresponds to a specific algorithm in the same way that each quantum gate array circuit is an implementation of a specific algorithm. To signal whether the computation has been completed, the processor contains a special internal qubit, $|n_0 \stackrel{\text{def}}{=} h\rangle$, known as the halt qubit, that is initialized to 0 and is set to 1 upon completion of the computation. Thus, an external operator (or classical computer) may periodically observe $|h\rangle$ to determine whether the computation has been completed. The evolution of the QTM can thus be described as

$$|\psi(s\Delta T)\rangle = U^s|\psi(0)\rangle,$$

where $|\psi(0)\rangle$ is the initial state, s is the number of computation steps, and ΔT is the time duration of each computational step.

3.2 The Universal Quantum Turing Machine

As Shi [32] pointed out, a UQTM state may be defined as $|Q, D, P, \Sigma\rangle$, where Q is the state of the processor, including the head position x, D is the state of the data register, and P is the program state. D and P are each parts of the tape and Σ is the remaining part of the tape that is not used during the computation. Note that this does not deviate from the original definition of the UQTM by Deutsch in [21], as the

corresponding basis elements of $|\mathbf{m}\rangle$ can be appropriately mapped to the corresponding basis elements of D, P, and Σ.

Deutsch claimed that there is a UQTM with which is associated a special unitary transformation U that when applied a positive integer number of times can come arbitrarily close to applying any desired unitary transformation on a finite number of data qubits. Stated differently, the claim was that there exists a UQTM, i.e. a special U, so that for an arbitrary accuracy ϵ and arbitrary unitary transformation $\mathcal{U}$ which changes D to $\mathcal{U}D$, there is always a program state $P(D,\mathcal{U},\epsilon)$ and a positive integer $s = s(D,\mathcal{U},\epsilon)$, such that

$$U^s|Q, D, P, \Sigma\rangle = |Q', D', P', \Sigma\rangle,$$

where D' is arbitrarily close to $\mathcal{U}D$, i.e. $\|D' - \mathcal{U}D\|^2 < \epsilon$. Finally, like the QTM, the UQTM contains a special internal halt qubit $|h\rangle$ that is monitored to determine whether the computation has completed.

3.3 Is The Halting Scheme Valid?

In 1997 Myers [22] suggested that the UQTM's halting scheme was invalid. He argued that an entanglement between the halt qubit and other qubits could occur, thereby making it impossible to determine whether the machine has halted or not. His reasoning was as follows: Suppose that two computations, A and B, halt after N_A and N_B steps, respectively, and without loss of generality, that $N_B > N_A$. Then for a computation that is a superposition of computations A and B, after N steps of the UQTM with $N_A < N < N_B$, the halt qubit will be in a superposition of halted and not halted states due to the linearity of the quantum evolution.

Because the computation time is unknown *a priori*, measurement of the halt qubit would collapse the state of the machine to that corresponding to the intermediate computation state of B (with $|h\rangle = |0\rangle$) or to the completed computation state of A (with $|h\rangle = |1\rangle$). Myers argued that this was a conflict between being universal and "being fully quantum," i.e. that the UQTM halting scheme was incompatible with superposition and hence the machine would need to operate on classic states. Conceptually, one could argue that this is not really a problem because any program will ultimately generate a single result. The case of superposed programs corresponds to the classical

case of running a program with random data. The computation result depends on the data. In the superposed quantum computer case, the result obtained depends on the final measurement probabilities for obtaining each of the superposed program results.

In 1998 Ozawa [33] showed that monitoring $|h\rangle$ is a quantum non-demolition measurement, that is, periodic measurement of $|h\rangle$ while the computation is in progress does not alter the final measurement of the memory tape contents, which store the result of the computation. This is true even if $|h\rangle$ becomes entangled with other qubits during the computation. The crucial aspect of this proof is that the probabilities of obtaining each of the possible superposed results is not altered by periodic measurement of the halt qubit. The periodic measurement could be said to collapse the machine to one of the many superposed branches of computation as Myers aptly highlighted, but the probability of measuring that particular computational branch is no different than if the measurement is postponed until after the program has completed execution. The key assumption or requirement in Ozawa's proof is that the state of the memory tape remain unchanged once the halt qubit is set.

Furthermore, in 2001 Shi [32] also highlighted that universality and "being fully quantum" does not require the *entire* UQTM to evolve from a superposition. The superposition need only be on the *data* state. For example, if the data state is $|D\rangle = |A\rangle + |B\rangle$, the state of the total system starts at $|Q, A+B, P(A+B,\mathcal{U},\epsilon), \Sigma\rangle$, rather than at $|Q, A, P(A,\mathcal{U},\epsilon), \Sigma\rangle + |Q, B, P(B,\mathcal{U},\epsilon), \Sigma\rangle$.

However, the scenario highlighted by Myer would arise if one were to require that the program be only dependent on the desired transformation $\mathcal{U}$ and the accuracy ϵ, but independent of the initial data state. In this case, a computation on data state $D = A+B$ would need to start at $|Q, A+B, P(\mathcal{U},\epsilon), \Sigma\rangle$, or $|Q, A, P(\mathcal{U},\epsilon), \Sigma\rangle + |Q, B, P(\mathcal{U},\epsilon), \Sigma\rangle$. Hence in this case entanglement between the halt qubit and the rest of the system would occur if the execution times for A and B were different, which would be generally the case. However, the requirement for a data state independent program is unnecessary and the halt qubit entanglement problem could thus be avoided. And if we require the programs to be data state independent and the halt qubit becomes entangled, Ozawa's proof applies and periodic measurements of the halt qubit do not affect the outcome of the computation.

However, Shi also pointed out that the halting scheme is a special case of the program *concatenation* scheme that was assumed to be valid in the original UQTM

proposal. The original definition of the UQTM is based on the assumption that if there is a program whose effect is to apply $\mathcal{U}$ on the data state $|D\rangle$, then there exists a unitary operator whose effect is $|h = 1\rangle\langle h = 0| \otimes \mathcal{U}$ on $|h = 0\rangle|D\rangle$. This assumption was not proven and the validity of program concatenation has not been addressed in other work (see section 8.3 in [34], for example) that relies upon the QTM defined by Bernstein and Vazirani [35] in 1997; this version of the QTM not only requires a halting scheme like Deutsch's but also requires that every computational path reach a final configuration simultaneously, and thus every computational path must be somehow synchronized.

The problem with synchronizing every computational path is that, in general, it is not known *a priori* how long a program will take to halt or if it will halt at all because program execution times can depend on the data that the program operates upon. This problem was highlighted by several authors, including Iriyama, Miyadera, and Ohya as recently as 2008 [36]. Thus, it is not always possible to find an upper bound T on the time needed for all branches to halt and thereby equip each branch of a computation with a counter that increments at each time step and halts once it reaches some upper bound T. In essence, such a synchronization scheme is well suited for dealing with sequential programs that are guaranteed to halt but not for programs that may never halt due to conditional branches or loops.

We address these open questions by constructing a theoretical universal quantum computer with valid and explicit halting and program concatenation schemes, and which also supports conditional branching and does not require synchronization of all computational paths. This machine serves as a prototypical model for a general-purpose programmable quantum computer that will be useful in the development and analysis of new quantum algorithms, complexity analysis of quantum algorithms, and investigation of the physical basis of the Turing halting problem.

3.4 A Universal Quantum Computer

Our goal is to devise a quantum computer that can compute any computable function. The machine itself is to be fixed and each different function is to be computed by providing the machine with a suitable set of input data and program. Any unitary operation can be approximated to any desired accuracy using the set of $\{\mathrm{H}, \mathrm{CNOT}, \mathrm{T}\}$

gates (see Nielsen & Chuang [1] chapter 4, for example), where

$$\begin{aligned} \mathrm{H} &\stackrel{\text{def}}{=} \frac{\mathrm{X}+\mathrm{Z}}{\sqrt{2}} = \frac{1}{\sqrt{2}}\left\{(|0\rangle + |1\rangle)\langle 0| + (|0\rangle - |1\rangle)\langle 1|\right\}, \\ \mathrm{CNOT} &\stackrel{\text{def}}{=} |00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 10| + |10\rangle\langle 11|, \\ \mathrm{T} &\stackrel{\text{def}}{=} |0\rangle\langle 0| + e^{i\pi/4}|1\rangle\langle 1|. \end{aligned}$$

This set is universal in the sense that any function (i.e. unitary operation) that can be computed by a quantum computer can be implemented using a combination of these gates. Thus, to create a universal quantum computer in the programmable sense, it suffices to devise one that can implement these operations on a specified set of qubits under the control of a quantum program. The quantum computer described below and illustrated in Figure 3.1 is an instance of such a machine.

Following Deutsch [21], our machine $\mathcal{UQC}$ consists of two primary parts: a processor $\mathcal{Q}$ that implements the universal set of unitary operations and an infinite tape that acts as the machine's external memory. The tape consists of an infinite sequence of qubits, $|M\rangle = \{|m_i\rangle\}, i \in \mathbb{Z}$, with only a finite subset of them ever being used for any given program. This corresponds to a classical computer's memory and external storage which, while finite, can be arbitrarily large. With the tape is associated an observable $\hat{x}$ in the processor that has the whole of $\mathbb{Z}$ as its spectrum and that acts as the address number of the currently scanned tape location. Addressing different tape locations can be realized either by a movable head over a static tape or by a movable tape under a static head. Since either scheme is identical for the purposes of constructing $\mathcal{UQC}$, we assume the latter as that allows for $\mathcal{Q}$ to be fixed in space, and movement of the tape is accomplished by a sliding 'bin' of qubits that moves under $\mathcal{Q}$'s control.

As part of its internal state machine, $\mathcal{Q}$ also contains two additional observables, $\hat{D}$ and $\hat{P}$, that act as the data address and program counter, respectively. $\hat{D}$ is used to address individual data qubits on the tape and to specify the branch destination address and $\hat{P}$ is used to keep track of the program instruction that is to be executed. As with classical computers, $\hat{D}$ and $\hat{P}$ need not have an infinite spectrum as they need only be as 'wide' as required to address the finite subset of the infinite tape that would ever be used. However, for the purpose of the most general construction, we do not restrict $\mathcal{UQC}$ to have a particular address range and thus treat $\hat{D}$ and $\hat{P}$ (and $\hat{x}$) as having an infinite spectrum.

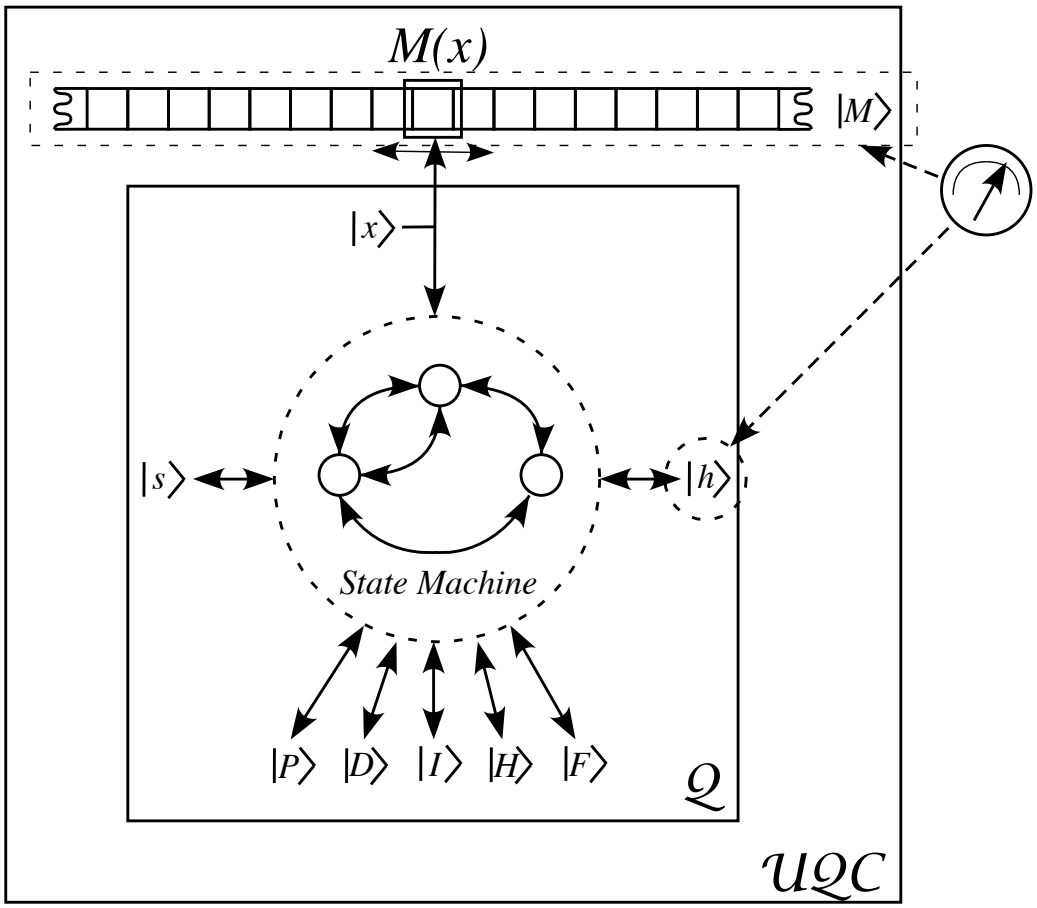

Figure 3.1: Universal Quantum Computer - Architecture of the universal quantum computer $\mathcal{UQC}$, showing the memory tape M, processor $\mathcal{Q}$, address of tape head $|x\rangle$, scratch qubit $|s\rangle$, instruction register $|I\rangle$, program address register $|P\rangle$, data address register $|D\rangle$, history address register $|H\rangle$, flow control register $|F\rangle$, halt qubit $|h\rangle$, and the qubits that are measured ($|M\rangle$ and $|h\rangle$).

$\mathcal{Q}$ also contains a 4-qubit register $\hat{I}$ to load the instruction to be executed. In order to perform the two-qubit CNOT operation, $\mathcal{Q}$ contains a 'scratch' qubit $|s\rangle$ that is used as the control qubit. Like Deutsch's UQTM, $\mathcal{UQC}$ also contains a dedicated observable qubit $|h\rangle$ that indicates whether the program execution has completed (i.e. the halt qubit). $\mathcal{Q}$ also contains a 2-qubit register $\hat{F}$ that is used to control the execution flow (i.e. whether the program should loop on the current instruction, proceed to the next instruction, or branch to a new instruction). Finally, $\mathcal{UQC}$ contains a register $\hat{H}$ with the same spectrum as $\hat{x}$, $\hat{D}$, and $\hat{P}$. The purpose (and naming) of the $\hat{H}$ register is described later. For notational simplicity, we drop the $\hat{}$ notation hereafter when referring to $\mathcal{UQC}$ registers, e.g. D refers to the observable $\hat{D}$ whose corresponding state is $|D\rangle$.

The overall state of $\mathcal{UQC}$, then, is given by $|h, x, D, P, F, H, s, I, M\rangle$, where $|h, D, P, F, H, s, I\rangle$ corresponds to Deutsch's $|\mathbf{n}\rangle$ with $|h\rangle \stackrel{\text{def}}{=} |n_0\rangle$.

Each program consists of a finite sequence of 4 qubit instruction words. Self-modifying code is to be avoided because modifying program instructions during program execution can lead to unpredictable results. For example, the processor fetches instructions to be executed from the memory tape into the temporary internal buffer register I by swapping the contents of the memory tape and the I register (and swapping back the two when the instruction has been executed). Because I is initialized to $|0\rangle$, the swapped contents of the memory tape temporarily become $|0\rangle$ while the instruction is being executed. This means that if the program attempts to modify the location of the instruction being executed, it would be modifying $|0\rangle$ and not the actual instruction (that is temporarily held in the I register). This can lead to unintended and unpredictable behavior.

The instruction set of $\mathcal{UQC}$ is as follows. As mentioned earlier, we implement a universal set of unitary operations, namely $\{\mathrm{H}, \mathrm{CNOT}, \mathrm{T}\}$, in order to ensure that $\mathcal{UQC}$ is universal. In order to enable the programmer to address any qubit on the memory tape and thus apply the universal set of operations to any qubit, we implement three instructions: An instruction to set D to 0, an instruction to increment D by 1, and an instruction to decrement D by 1. Because the CNOT operation requires two operands (control and data), we implement a swap instruction to enable the programmer to swap the qubit on the memory tape pointed to by D with the machine's s qubit, thereby enabling any qubit on the memory tape to be used as the control qubit. While not

strictly necessary for universality, we implement a branching scheme in $\mathcal{UQC}$ because firstly, this is not explicitly possible in other popular quantum computing frameworks such as the gate array framework and secondly, because it is a common operation in classical computers. Branching is essentially implemented by allowing the programmer to swap the data register and program counter register contents, thereby allowing the program to branch to any instruction on the memory tape. We also implement an instruction to effectively clear s by swapping its contents with the next available 0 slot on the negative portion of the memory tape (pointed to by H). The clear s instruction provides for a simple and convenient way for the programmer to load s with 0 without having to hunt around the memory tape looking for a 0 data qubit slot. Finally, we implement an instruction to set the halt qubit to 1 but because we also want the memory tape to remain unchanged once the halt qubit is set, we implement an accompanying instruction (NOP) to follow the halt instruction that will accomplish this.

The instruction set of $\mathcal{UQC}$, then, consists of 11 instructions, whose operations and encodings are defined in Table 4.1. The single qubit operations H and T act on the qubit at tape location $M(D)$, denoted $|M\rangle_D$, and the two qubit operations SWAP and NAND act on $|M\rangle_D$ and the scratch qubit $|s\rangle$, the latter being used as the control qubit for the NAND operation.

Table 3.1: $\mathcal{UQC}$ Instruction Set

Label	Encoding	Description
$\|\text{NOP}\rangle$	$\|0000\rangle$	No operation
$\|\text{D} \to 0\rangle$	$\|0001\rangle$	$D \to 0$
$\|\text{D} + 1\rangle$	$\|0010\rangle$	$D \to D + 1$
$\|\text{D} - 1\rangle$	$\|0011\rangle$	$D \to D - 1$
$\|\text{H}\rangle$	$\|0100\rangle$	Apply Hadamard operation to $\|M\rangle_D$
$\|\text{T}\rangle$	$\|0101\rangle$	Apply T operation to $\|M\rangle_D$
$\|\text{SWAP}\rangle$	$\|0110\rangle$	$\|M\rangle_D \leftrightarrow \|s\rangle$
$\|\text{CNOT}\rangle$	$\|0111\rangle$	CNOT of $\|M\rangle_D$ and $\|s\rangle$ ($\|s\rangle$: control)
$\|\text{D} \leftrightarrow \text{P}\rangle$	$\|1000\rangle$	$\|D\rangle \leftrightarrow \|P\rangle$ (branch) iff $s = 0$
$\|\text{CLS}\rangle$	$\|1001\rangle$	Clear s
$\|\text{h} \to 1\rangle$	$\|1111\rangle$	$\|h\rangle \to \|1\rangle$ (set halt qubit)

The operation of $\mathcal{UQC}$ proceeds as follows:

1. An external operator (or classical computer) initializes the state of M at $t = 0$ with the desired data and program. Data qubit i, $i \in \mathbb{Z}^+$, is placed on tape location $M(5i-1)$ and program instruction j, $j \in \mathbb{Z}^+$, is placed on tape locations $M(5j-2:5(j-1))$, i.e. data is placed at $M(4)$, $M(9)$, $M(14)$, ..., and program instructions are placed at $M(3:0)$, $M(8:5)$, $M(18:15)$, ... The negative portions of the tape are initialized to $|0\rangle$, as illustrated in 3.2.

2. The processor's H register is initialized to $|-1\rangle$ and all other registers are initialized to $|0\rangle$. This is the so-called "reset state" of $\mathcal{Q}$.

3. An external operator starts $\mathcal{Q}$ by releasing it from its reset state.

4. $\mathcal{Q}$ fetches the program instruction at tape location $M(P)$ into register I.

5. $\mathcal{Q}$ executes the operation specified by I.

6. If the halt qubit $|h\rangle$ becomes set, $\mathcal{Q}$ halts execution (strictly speaking, because $\mathcal{UQC}$ is a quantum system, $\mathcal{Q}$ continues to evolve but the evolution of the memory tape becomes trivial – i.e. $U = \mathbb{1}$ – after the halt qubit has been set) and awaits an external measurement of the results. Otherwise, $\mathcal{Q}$ continues execution of the program by loading the next program instruction.

7. An external operator periodically performs a measurement on the halt qubit.

8. If measurement of the halt qubit yields $|1\rangle$, the program has completed execution. The results are obtained by measuring the contents of M. Otherwise, $\mathcal{Q}$ is allowed to continue program execution.

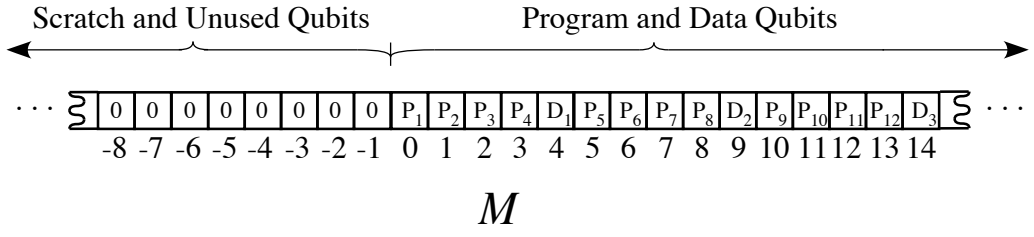

Figure 3.2: Memory Tape Diagram - Initial memory tape contents. The negative qubit slots are used as scratch qubits and the non-negative qubit slots are initialized with interleaved program instruction and data qubits.

The operation of $\mathcal{Q}$ is governed by the state machine depicted in Figure 3.3.

3.4.1 The Evolution of $\mathcal{Q}$

We now define the unitary evolution operators associated with the $\mathcal{Q}$ state transitions. In the equations below, subscripts on projectors denote the qubit(s) on which the projector acts, e.g. $|i\rangle\langle i|_k$ acts on qubit k, and unspecified qubits are understood to be operated on by an implicit identity operator, e.g. $|i\rangle\langle j|_k \otimes |l\rangle\langle m|_n$ is short hand for $|i\rangle\langle j|_k \otimes |l\rangle\langle m|_n \otimes \mathbb{1}_{\neq k,n}$ which acts on qubits k and n and leaves all other qubits unaffected. $|\psi\rangle_R$ denotes $\prod_i |\psi(i)\rangle_{R(i)}$, where R is a multiple qubit register (e.g. D) with $R = \prod_i R(i)$ and $\psi = \prod_i \psi(i)$. Moreover, for notational simplicity in the rest of this paper, we define 4 primitive unitary operations, SWAP, DEC, INC, and CNOT as follows:

1. Swap contents of registers a and b

$$\mathrm{SWAP}_{a,b} \stackrel{\mathrm{def}}{=} \sum_{i,j} |i\rangle\langle j|_a \otimes |j\rangle\langle i|_b \tag{3.1}$$

2. Decrement the contents of register a

$$\mathrm{DEC}_a \stackrel{\mathrm{def}}{=} \sum_i |i-1\rangle\langle i|_a \tag{3.2}$$

3. Increment the contents of register a

$$\mathrm{INC}_a \stackrel{\mathrm{def}}{=} \sum_i |i+1\rangle\langle i|_a \tag{3.3}$$

4. CNOT operation using qubit a as the control and qubit b as the data

$$\mathrm{CNOT}_{a,b} \stackrel{\mathrm{def}}{=} (|00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 10| + |10\rangle\langle 11|)_{a,b} \tag{3.4}$$

The operators, U_{IF},U_{XD}, U_{EX}, U_{UD}, U_{UF}, U_{RI}, U_{UP}, U_{CF}, that govern the $\mathcal{Q}$ state transitions, then, are defined as follows.

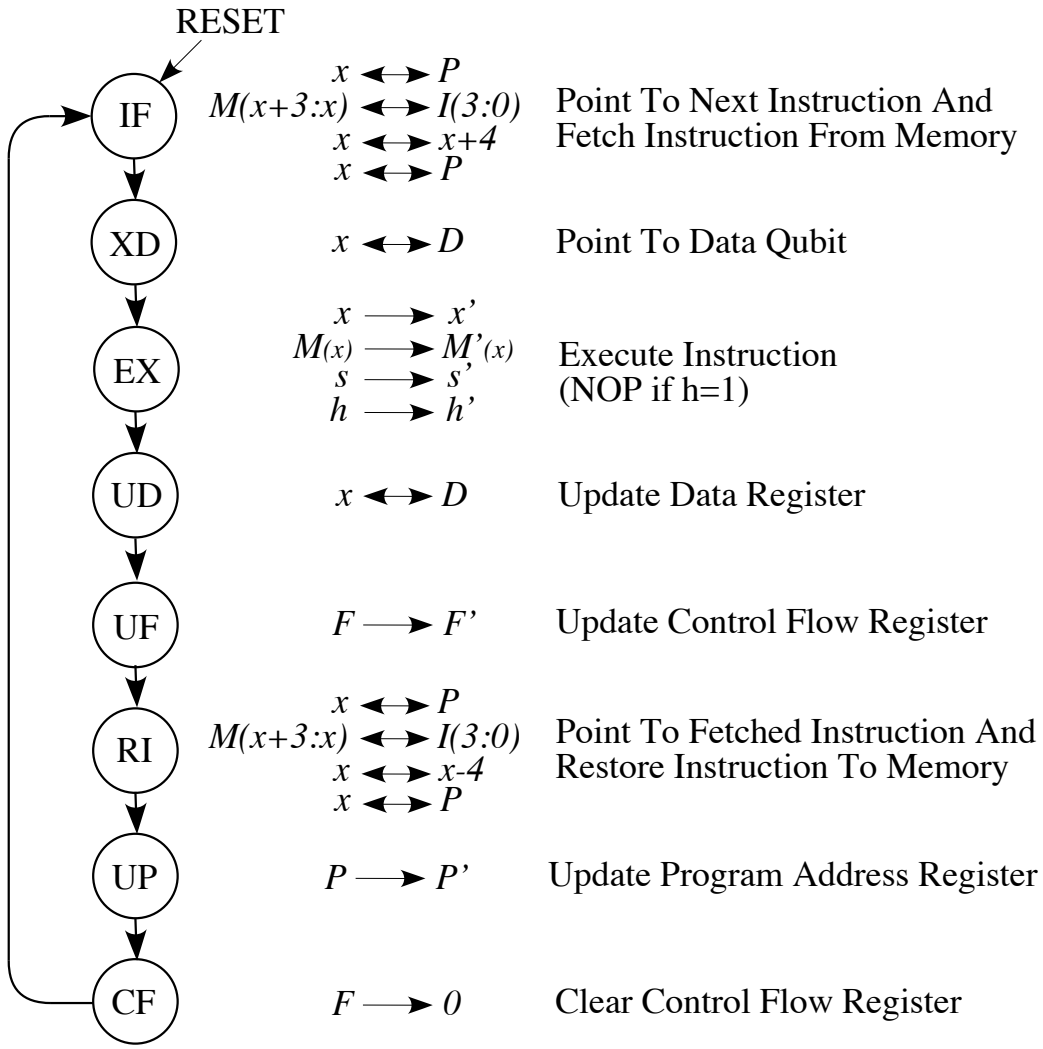

Figure 3.3: State Machine Diagram - $\mathcal{Q}$ state machine diagram that corresponds to the evolution of the universal quantum computer. The overall evolution is determined by eight unitary transformations.

1. Fetch next instruction at $M(P)$

$$U_{\mathrm{IF}} \stackrel{\mathrm{def}}{=} \mathrm{DEC}_P^4 \cdot \mathrm{SWAP}_{x,P} \cdot \Big(\prod_{i=3}^{0} \mathrm{INC}_x \cdot \mathrm{SWAP}_{M(x),I(i)} \Big) \cdot \mathrm{SWAP}_{x,P} \tag{3.5}$$

This operator fetches the next program instruction by 'swapping' the next program instruction qubits on the memory tape with the contents of the I register. As stated earlier, because I is initialized to $|0\rangle$, the instruction slot on the memory tape becomes temporarily $|0\rangle$ while the instruction is being executed but is restored to its original state once the instruction has been executed. Note that P will be pointing back to the fetched instruction address after this operator is applied because the update of the program counter is deferred until U_{UP} is applied.

2. Move tape head to $M(D)$

$$U_{\mathrm{XD}} \stackrel{\mathrm{def}}{=} \mathrm{SWAP}_{x,D} \tag{3.6}$$

U_{XD} points the memory tape head to the qubit addressed by the data register D.

3. Execute instruction

$$\begin{aligned} U_{\mathrm{EX}} \stackrel{\mathrm{def}}{=} & |\mathrm{NOP}\rangle\langle\mathrm{NOP}|_I \\ & + |\mathrm{D} \to 0\rangle\langle\mathrm{D} \to 0|_I \otimes \mathrm{DEC}_x \\ & + |\mathrm{D}+1\rangle\langle\mathrm{D}+1|_I \otimes \mathrm{INC}_x \\ & + |\mathrm{D}-1\rangle\langle\mathrm{D}-1|_I \otimes \mathrm{DEC}_x \\ & + |\mathrm{H}\rangle\langle\mathrm{H}|_I \otimes \mathrm{H}_{M(x)} \\ & + |\mathrm{T}\rangle\langle\mathrm{T}|_I \otimes \mathrm{T}_{M(x)} \\ & + |\mathrm{SWAP}\rangle\langle\mathrm{SWAP}|_I \otimes \mathrm{SWAP}_{M(x),s} \\ & + |\mathrm{CNOT}\rangle\langle\mathrm{CNOT}|_I \otimes \mathrm{CNOT}_{s,M(x)} \\ & + |\mathrm{D} \leftrightarrow \mathrm{P}\rangle\langle\mathrm{D} \leftrightarrow \mathrm{P}|_I \\ & + |\mathrm{CLS}\rangle\langle\mathrm{CLS}|_I \otimes \mathrm{SWAP}_{x,H} \cdot \mathrm{DEC}_x \cdot \mathrm{SWAP}_{M(x),s} \cdot \mathrm{SWAP}_{x,H} \\ & + |\mathrm{h} \to 1\rangle\langle\mathrm{h} \to 1|_I \otimes \big(|1\rangle\langle 0| + |0\rangle\langle 1| \big)_h \\ & + \sum_{i=0}^{4} |\mathrm{R_i}\rangle\langle\mathrm{R_i}|_I \end{aligned} \tag{3.7}$$

U_{EX} applies the appropriate transformation associated with the instruction being executed. The transformations associated with the NOP, D + 1, D − 1, H, T, SWAP, CNOT, and reserved instructions are self evident but those associated with the D → 0, D ↔ P, CLS, and h → 1 instructions warrant some explanation.

The D $\rightarrow$ 0 instruction works as follows. D is decremented by one by U_{EX} and P is left unchanged by U_{UP} (see equation (3.11)) until $D = 0$. Leaving the program counter unchanged has the effect of keeping P pointing to the D $\rightarrow$ 0 instruction such that it is re-fetched in the next iteration. Thus, $\mathcal{Q}$ continues to fetch and execute the same D $\rightarrow$ 0 instruction until $D = 0$. In other words, it will loop on the D $\rightarrow$ 0 instruction, decrementing D until it reaches 0. Once $D = 0$, P is incremented by 5 such that it points to the next instruction, thus completing the loop.

It is important to note that this scheme relies on the assumption that $D > 0$ when the D $\rightarrow$ 0 instruction is encountered. Therefore, the programmer must ensure that $D > 0$ when $\mathcal{Q}$ fetches the D $\rightarrow$ 0 instruction. This can be accomplished by preceding the D $\rightarrow$ 0 instruction with a D + 1 instruction since, in the absence of programming error, D will always be positive. If the programmer fails to meet this requirement $\mathcal{Q}$ could loop forever stepping through the negative portions of the memory tape.

The transformation associated with the D $\leftrightarrow$ P operation is the identity operation here because its execution is deferred until later. Deferring the actual swapping of the D and P register contents is necessary in order to keep the address of the D $\leftrightarrow$ P instruction unchanged so that we can restore the branch instruction back to its original slot on the memory tape and only then update the program counter to point to the next instruction in the program execution flow.

The CLS instruction first points the memory tape head to the address contained in the H register (the next slot on the negative portion of the memory tape that contains $|0\rangle$), swaps the contents of the s qubit with the contents of the memory tape slot ($|0\rangle$) thereby clearing s (but leaving the previous value of s on the memory tape making the operation reversible in principle), decrements H such that it points to the next $|0\rangle$ slot on the memory tape, and then points the memory tape head back to where it was.

The NOP instruction plays a key role in the $\mathcal{UQC}$ halting scheme. In our implementation, the halting scheme requires the halt instruction $|\mathrm{h} \rightarrow 1\rangle$ to be followed by a $|\mathrm{NOP}\rangle$ instruction. In other words, the 'true' halt instruction is effectively $|\mathrm{h} \rightarrow 1\rangle|\mathrm{NOP}\rangle$ or $|11110000\rangle$ using the presently defined instruction encodings.

This is such that after the halt qubit is set, $\mathcal{Q}$ will continue to fetch the next instruction following $|\mathrm{h} \rightarrow 1\rangle$ which being $|\mathrm{NOP}\rangle$ will guarantee that $\mathcal{Q}$ loops forever doing nothing, thereby effectively halting program execution (but not quantum evolution). The fact that the encoding of the NOP instruction is $|0000\rangle$ is also intentional. This ensures that the contents of the memory tape remain unchanged after the halt qubit is set because swapping the instruction slot on the memory tape with the contents of the I register leaves the state of the memory tape unchanged. The halting scheme relies on all halt instructions in any given program being followed by a NOP instruction and stopping the program counter from changing when a NOP instruction is executed such that P will continue to point at the NOP instruction following the instruction that caused the halt qubit to be set.

The halting scheme is thus effectively a two step process: the first step is to set the halt qubit using the $\mathrm{h} \rightarrow 1$ instruction to alert an external observer that the program has halted and the second step is to loop forever on the NOP instruction. In this sense, the NOP instruction is really a "loop forever" trap instruction. As such, the NOP instruction must only be used following a halt instruction. If it is inadvertently placed anywhere else in the program, program execution will halt but the halt qubit will not be set so the external observer will not know that the program has halted.

An improved halting scheme that does not require all instances of the halt instruction in a program to be followed by a NOP instruction may be possible and is an area for future investigation.

4. Update contents of D register

$$U_{\mathrm{UD}} \stackrel{\mathrm{def}}{=} \mathrm{SWAP}_{x,D} \tag{3.8}$$

U_{UD} updates the D register with the results of executing the instruction since x will contain any changes to D after U_{EX} has been applied.

5. Update control flow register with instruction flow information

$$\begin{aligned} U_{\mathrm{UF}} \stackrel{\mathrm{def}}{=} & \big(|\mathrm{D} \rightarrow 0\rangle\langle \mathrm{D} \rightarrow 0|_I \otimes (\mathbb{1} - |0\rangle\langle 0|)_D + |\mathrm{NOP}\rangle\langle \mathrm{NOP}|_I\big) \\ & + |\mathrm{D} \leftrightarrow \mathrm{P}\rangle\langle \mathrm{D} \leftrightarrow \mathrm{P}|_I \otimes |0\rangle\langle 0|_s \otimes \mathrm{INC}_F^2 \\ & + \big(\mathbb{1} - |\mathrm{D} \rightarrow 0\rangle\langle \mathrm{D} \rightarrow 0|_I \otimes (\mathbb{1} - |0\rangle\langle 0|)_D - |\mathrm{NOP}\rangle\langle \mathrm{NOP}|_I \\ & \quad - |\mathrm{D} \leftrightarrow \mathrm{P}\rangle\langle \mathrm{D} \leftrightarrow \mathrm{P}|_I \otimes |0\rangle\langle 0|_s\big) \otimes \mathrm{INC}_F \end{aligned} \tag{3.9}$$

Table 3.2: F Encodings

Label	Encoding	Description
$\|\text{LOOP}\rangle$	$\|00\rangle$	Loop ($P \rightarrow P$)
$\|\text{NEXT}\rangle$	$\|01\rangle$	Next Sequential Instruction ($P \rightarrow P + 5$)
$\|\text{BR2D}\rangle$	$\|10\rangle$	Branch to D ($P \rightarrow D$)
$\|\text{R}_0\rangle$	$\|11\rangle$	Unused

U_{UF} updates F whose value is later used to update P to point to the address of the next instruction to be executed. Note that F is initialized to $|0\rangle$ and the evolution of $\mathcal{UQC}$ is designed to ensure that $F = 0$ when U_{UF} is applied (F is effectively "cleared" by U_{CF} by swapping its contents with the infinite supply of $|0\rangle$ slots on the negative portion of the memory tape as we describe later). As explained earlier, if the instruction is $\text{D} \rightarrow 0$ and $D \neq 0$ or if the instruction is NOP, P will be left unchanged to effectively loop on the instruction. If the instruction is $\text{D} \leftrightarrow \text{P}$ and $s = 0$ then P will be swapped with D to effectively branch to D. Otherwise, P is set to point to the instruction following the instruction that was just executed (i.e. $P \rightarrow P + 5$). The encodings of F are defined in Table 3.2.

6. Restore executed instruction back to the memory tape location from where it was fetched

$$U_{\text{RI}} \stackrel{\text{def}}{=} U_{\text{IF}}^{\dagger} \tag{3.10}$$

U_{RI} restores the instruction that was just executed back to its original slot on the memory tape. Recall that the P update has been deferred and will be controlled by the state of the F register. Thus, the only operator that has affected P thus far has been U_{IF} so $U_{\text{IF}}^{\dagger}$ suffices to undo the fetch. In essence, F is a temporary place holder to store the information necessary to determine the next instruction location after restoring the instruction back to the memory tape and hence losing knowledge of how to update P otherwise.

7. Update program counter to the address of the next instruction to be executed

$$\begin{aligned} U_{\text{UP}} \stackrel{\text{def}}{=} & |\text{LOOP}\rangle\langle\text{LOOP}|_F \\ & + |\text{NEXT}\rangle\langle\text{NEXT}|_F \otimes \text{INC}_P^5 \\ & + |\text{BR2D}\rangle\langle\text{BR2D}|_F \otimes \text{SWAP}_{D,P} \\ & + |\text{R}_0\rangle\langle\text{R}_0|_F \end{aligned} \tag{3.11}$$

U_{UP} updates P to the address of the next instruction to be executed according to the state of F.

8. Clear flow control register such that it can be used again in the next cycle

$$U_{\mathrm{CF}} \stackrel{\mathrm{def}}{=} \mathrm{SWAP}_{x,H} \cdot \Big(\prod_{i=1}^{0} \mathrm{DEC}_x \cdot \mathrm{SWAP}_{M(x),F(i)} \Big) \cdot \mathrm{SWAP}_{x,H} \tag{3.12}$$

U_{CF} first swaps the contents of the x and H registers. The H register contains the address of the next slot on the negative portion of the tape that contains $|00\rangle$. These '0' slots are used to clear the F register back to $|0\rangle$ each cycle. Since the sequence of F values effectively contains the information about the program execution flow, in essence the negative portion of the tape contains the 'history' of instructions that $\mathcal{UQC}$ has executed and is a side-effect of the need for all $\mathcal{UQC}$ programs to be reversible.

In other words, the negative portion of M is used to store the ancillary garbage data that would be required to reverse the operation of the program. The number of $|0\rangle$ slots required for any given program is equal to the number of instructions that are executed by the program. U_{CF} clears F by swapping its contents with the contents of the next $|0\rangle$ slot on the negative portion of the tape. After application of U_{CF}, H points to the next $|0\rangle$ slot on the tape and the previous F value is contained on the slot to the right of the first $|0\rangle$ slot on the negative portion of the tape.

At this point, $\mathcal{Q}$ has completed processing the instruction and is ready to fetch the next instruction in the execution flow.

These operators are all readily verified to be unitary and to have the desired effects of implementing the state machine shown in Figure 3.3. The overall evolution of $\mathcal{UQC}$, then, is governed by the unitary operator,

$$U = U_{\mathrm{CF}} U_{\mathrm{UP}} U_{\mathrm{RI}} U_{\mathrm{UF}} U_{\mathrm{UD}} U_{\mathrm{EX}} U_{\mathrm{XD}} U_{\mathrm{IF}}. \tag{3.13}$$

Unlike Deutsch's original UQTM, the memory tape (or tape head) of $\mathcal{UQC}$ is not restricted to move at most one position to the left or to the right ($x \to x \pm 1$) in any given step. This is most obvious in the case of the branch instruction where the tape head will jump by an arbitrarily large amount in a single step. However, the evolution of $\mathcal{UQC}$ is still unitary and hence physically possible in principle.

3.4.2 Illustration of program execution

Since the overall evolution of $\mathcal{U}$ is moderately complex, we illustrate how $\mathcal{U}$ operates by stepping through a simple program that swaps the first two data qubits on the memory tape (qubits $D(1)$ and $D(2)$) in Appendix A. As can be seen in the execution trace, once the halt qubit has been set, the state of the memory tape is the desired result with the first two data qubits swapped and all other data qubits on the tape unchanged and no further changes to the memory tape occur. Strictly speaking, even after the halt qubit has been set, the scratch qubits on the tape will be forever swapped with the contents of the I register but because the I register contains the NOP whose encoding is 0 and the scratch qubits are initialized to $|0\rangle$, the net effect is that the state of the qubits on the memory tape will remain unchanged once the halt qubit has been set.

3.5 Some Primitive Programs

In this section we describe a set of primitive programs or operations to demonstrate the universal nature of $\mathcal{UQC}$. These routines serve as building blocks for devising and analyzing more complicated and useful programs.

The first set of primitive programs, $\{|D_{+i}\rangle, |D_{-i}\rangle, |D_i\rangle, |S_{i,s}\rangle, |S_{i,j}\rangle, |B_i\rangle\}$, that we define perform basic functions to manipulate the data address register, swap qubits, and conditionally branch to an arbitrary address. The superscripts on the programs denote the operation performed by the program and the subscripts indicate the qubits on which the program operates. For notational simplicity, $|P_h\rangle$ denotes the program that causes $\mathcal{UQC}$ to halt, i.e. $|P_h\rangle \stackrel{\text{def}}{=} |\text{h} \rightarrow 1\rangle|\text{NOP}\rangle$.

1. $|D_{+i}\rangle$: Increment D by i

$$|D_{+i}\rangle \stackrel{\text{def}}{=} \begin{cases} \prod_{k=1}^{i} |\text{D} + 1\rangle, \text{ if } i \geq 1 \\ \mathbb{1}, \text{ otherwise} \end{cases} \tag{3.14}$$

2. $|D_{-i}\rangle$: Decrement D by i

$$|D_{-i}\rangle \stackrel{\text{def}}{=} \begin{cases} \prod_{k=1}^{i} |\text{D} - 1\rangle, \text{ if } i \geq 1 \\ \mathbb{1}, \text{ otherwise} \end{cases} \tag{3.15}$$

3. $|D_i\rangle$: Set D to i, $i > 0$

$$|D_i\rangle \stackrel{\text{def}}{=} |\mathrm{D}+1\rangle|\mathrm{D}\to 0\rangle|D_{+i}\rangle \tag{3.16}$$

Recall from the discussion of U_{EX}, that we are preceding the $\mathrm{D}\to 0$ instruction with a $\mathrm{D}+1$ instruction to ensure that $D > 0$ when the $\mathrm{D}\to 0$ instruction is executed.

4. $|S_{i,s}\rangle$: Swap data qubits $D(i)$ and s

$$|S_{i,s}\rangle \stackrel{\text{def}}{=} |D_{5i-1}\rangle|\mathrm{SWAP}\rangle \tag{3.17}$$

5. $|S_{i,j}\rangle$: Swap data qubits $D(i)$ and $D(j)$

$$|S_{i,j}\rangle \stackrel{\text{def}}{=} |S_{5i-1,s}\rangle|S_{5j-1,s}\rangle|S_{5i-1,s}\rangle \tag{3.18}$$

6. Branch to the i-th instruction (i.e. instruction at $M(5(i-1))$, where $i \in \mathbb{Z}^+$

$$|B_i\rangle \stackrel{\text{def}}{=} |D_{5(i-1)}\rangle|\mathrm{D}\leftrightarrow \mathrm{P}\rangle \tag{3.19}$$

Note that, as defined, this instruction will have no effect unless $|s\rangle = |0\rangle$ so this operation is only useful following non-trivial operations on the $|s\rangle$ qubit.

Next we describe a set of programs, $\{|P_i^{\mathrm{H}}\rangle, |P_{i,j}^{\mathrm{H}}\rangle, |P_i^{\mathrm{T}}\rangle, |P_{i,j}^{\mathrm{C}}\rangle\}$, to apply the H, T, and CNOT operations on arbitrary qubits i and j on the memory tape, where i and $j \in \mathbb{Z}$. These comprise a universal set of unitary operations from which any arbitrary unitary operation can be constructed.

1. $|P_i^{\mathrm{H}}\rangle$: Apply H to data qubit $D(i)$

$$|P_i^{\mathrm{H}}\rangle \stackrel{\text{def}}{=} |D_{5i-1}\rangle|\mathrm{H}\rangle \tag{3.20}$$

2. $|P_{i,j}^{\mathrm{H}}\rangle$: Apply H to data qubits $D(i:j)$, where $i \geq j$

$$|P_{i,j}^{\mathrm{H}}\rangle \stackrel{\text{def}}{=} \prod_{k=j}^{i} |P_k^{\mathrm{H}}\rangle \tag{3.21}$$

One could implement this program using a loop but that would require first implementing binary addition of M qubits. Binary addition is possible because

one can implement a binary adder such as a Carry Lookahead Adder (CLA) [37] using the NAND program that we define later in this section. However, since we are only interested in a polynomial order (in the number of qubits) multiple qubit Hadamard transformation program, we define $|P^{\mathrm{H}}_{i,j}\rangle$ as a sequential 'unrolled' loop program.

3. $|P^{\mathrm{T}}_i\rangle$: Apply T to data qubit $D(i)$

$$|P^{\mathrm{T}}_i\rangle \stackrel{\mathrm{def}}{=} |D_{5i-1}\rangle|\mathrm{T}\rangle \tag{3.22}$$

4. $|P^{\mathrm{C}}_{i,j}\rangle$: Apply CNOT to data qubits $D(i)$ and $D(j)$ with $D(i)$ as the control qubit

$$|P^{\mathrm{C}}_{i,j}\rangle \stackrel{\mathrm{def}}{=} |S_{i,s}\rangle|D_{5j-1}\rangle|\mathrm{CNOT}\rangle|S_{i,s}\rangle \tag{3.23}$$

Using the sets of primitive programs defined above, we can now define the set of programs, $\{|P^{\mathrm{X}}_i\rangle, |P^{\mathrm{S}}_i\rangle, |P^{\mathrm{T}^\dagger}_i\rangle\}$, that apply the Pauli X, Phase (S), and $\mathrm{T}^\dagger$ operations on data qubit $i \in \mathbb{Z}^+$. These operations are often used in quantum algorithms so it is useful to identify the programs that implement them. A constant subscript on a program denotes that some suitable qubit on the memory tape has been prepared with the appropriate value. For example, $|P_1\rangle$ is shorthand for $|P_k\rangle$ where $M(k)$, for some suitable k, has been prepared with the value $|1\rangle$.

1. $|P^{\mathrm{X}}_i\rangle$: Apply σ_x to qubit $M(i)$

$$|P^{\mathrm{X}}_i\rangle = |P^{\mathrm{C}}_{1,i}\rangle. \tag{3.24}$$

$|P^{\mathrm{X}}_i\rangle$: $|1\rangle_j|\psi\rangle_i \rightarrow |1\rangle_j|1 \oplus \psi\rangle_i = |1\rangle_j|\overline{\psi}\rangle_i = |1\rangle_j\sigma_x|\psi\rangle_i$.

2. $|P^{\mathrm{S}}_i\rangle$: Apply phase (S) to qubit $M(i)$

Noting that $\mathrm{S} = \mathrm{T}^2$, the following program implements the phase operation.

$$|P^{\mathrm{S}}_i\rangle = |P^{\mathrm{T}}_i\rangle|P^{\mathrm{T}}_i\rangle \tag{3.25}$$

$|P^{\mathrm{S}}_i\rangle$: $|\psi\rangle_i \rightarrow \mathrm{T}^2|\psi\rangle_i = \mathrm{S}|\psi\rangle_i$.

3. $|P_i^{\mathrm{T}^\dagger}\rangle$: Apply $\mathrm{T}^\dagger$ (reverse T) to qubit $M(i)$

   This operation is used to define the Toffoli operation in a later section so we define it here. Noting that $\mathrm{S} = \mathrm{T}^2$ and that $S^4 = \mathbb{1}$, $\mathrm{T}^\dagger = \mathrm{T}^\dagger\mathrm{S}^4 = \mathrm{T}^\dagger\mathrm{T}^2\mathrm{S}^3 = \mathrm{TS}^3$, the following program implements the $\mathrm{T}^\dagger$ operation.

$$|P_i^{\mathrm{T}^\dagger}\rangle = |P_i^{\mathrm{S}}\rangle|P_i^{\mathrm{S}}\rangle|P_i^{\mathrm{S}}\rangle|P_i^{\mathrm{T}}\rangle \tag{3.26}$$

Although not specifically shown here, other useful quantum gates such as σ_y, σ_z, entanglement gate, etc. can be similarly implemented. These enable us to implement any algorithm from the quantum gate array framework on $\mathcal{UQC}$ by appropriate combinations of the programs we have just defined and adding $|P_h\rangle$ as the last step in the combined program to halt $\mathcal{UQC}$ upon completion. Since the quantum gate array framework is universal (see [1] chapter 4, for example), this means that $\mathcal{UQC}$ is also quantum computationally universal with the additional advantage that $\mathcal{UQC}$ provides a fixed and programmable machine to implement the algorithms unlike the quantum gate array framework.

The two bit NAND operation is universal for classical computation. That is, the NAND operation can be used to implement any Boolean function. Hence it is useful to define a program that emulates the NAND operation on two qubits as this could be used as the basis for emulating classical functions on $\mathcal{UQC}$. For this purpose, we first define a program that implements the Toffoli operation which itself is a universal classical gate [1]. The Toffoli program, $|P_{i,j,k}^{\mathrm{Toff}}\rangle$, applies the Toffoli operation to qubits $D(i)$, $D(j)$, and $D(k)$, where $D(i)$ and $D(j)$ are the control qubits and $D(k)$ is the target qubit.

Armed with the Toffoli program, implementing a program that takes the NAND of qubits $D(i)$ and $D(j)$ and storing the result in qubit $D(c)$ is a simple matter of executing the program $|P_{i,j,c}^{\mathrm{NAND}}\rangle = |P_{i,j,1}^{\mathrm{Toff}}\rangle$.

$$|P_{i,j,k}^{\mathrm{Toff}}\rangle = |P_k^{\mathrm{H}}\rangle|P_{j,k}^{\mathrm{C}}\rangle|P_k^{\mathrm{T}^\dagger}\rangle|P_{i,k}^{\mathrm{C}}\rangle|P_k^{\mathrm{T}}\rangle|P_{j,k}^{\mathrm{C}}\rangle|P_k^{\mathrm{T}^\dagger}\rangle|P_{i,k}^{\mathrm{C}}\rangle|P_k^{\mathrm{T}}\rangle|P_j^{\mathrm{T}^\dagger}\rangle|P_{i,j}^{\mathrm{C}}\rangle|P_k^{\mathrm{H}}\rangle|P_j^{\mathrm{T}^\dagger}\rangle|P_{i,j}^{\mathrm{C}}\rangle|P_j^{\mathrm{S}}\rangle|P_i^{\mathrm{T}}\rangle \tag{3.27}$$

The ability to perform a two-qubit NAND operation gives $\mathcal{UQC}$ the ability to compute any classically computable function thus demonstrating that it can emulate a classical universal Turing machine. This is in addition to being a universal quantum

computer since it can also implement the set of universal quantum operations on arbitrary qubits on its memory tape as shown earlier. $\mathcal{UQC}$ can compute any classically computable function, it can compute any quantum computable function, and it is programmable. In short, $\mathcal{UQC}$ is computationally universal.

3.6 Program Concatenation Scheme

In the process of defining the primitive programs in the preceding section, we have implicitly used program concatenation whereby we sequentially combined separate programs to create larger programs. Strictly speaking, the programs that we have thus far defined are really subroutines since complete programs must include the halting program, $|P_h\rangle$, in order to signal program completion. However, it is readily seen that all of the primitive subroutines can be converted into full-fledged programs by adding $|P_h\rangle$ as the last instruction.

Sequential programs (programs without branch instructions) can thus be concatenated by simply removing the last $|P_h\rangle$ step from each constituent program, concatenating the resulting subroutines, and appending on $|P_h\rangle$ at the end. Suppose that we have two sequential programs, $|P_A\rangle$ and $|P_B\rangle$ that we wish to concatenate to create a program $|P_{AB}\rangle$ whose effect is to execute $|P_B\rangle$ followed by $|P_A\rangle$. Since $|P_A\rangle$ and $|P_B\rangle$ are sequential, this means that $|P_h\rangle$ is the last step in each program. That is, $|P_A\rangle = |P_{A'}\rangle|P_h\rangle$ and $|P_B\rangle = |P_{B'}\rangle|P_h\rangle$. Thus, to achieve the effect of running $|P_A\rangle$ followed by $|P_B\rangle$, we simply construct the program $P_{AB} \stackrel{\text{def}}{=} |P_{A'}\rangle|P_{B'}\rangle|P_h\rangle$.

The situation is quite different for branching programs. In general, without complete knowledge of the operations of the programs to be concatenated, it is not possible to concatenate them in the strictest sense of joining the individual programs into a single larger program. This is not a limitation of $\mathcal{UQC}$ but of any computer, be it classical or quantum. The problem is that the branch destinations in a branching program can be data dependent and the branching address may also be manipulated as data. Therefore, it is not sufficient to add an appropriate offset (the number of instructions of preceding concatenated programs) to all branch instructions because this would have the adverse effect of potentially adding an offset to the manipulated data and hence altering the intended computation results.

The solution, of course, is to first run $|P_A\rangle$, wait for it to complete, replace $|P_A\rangle$ with $|P_B\rangle$, reset the program counter register to 0, leave all other internal registers and memory tape qubits unchanged, and resume execution to run $|P_B\rangle$. However, strictly speaking, this is not program concatenation per se because while the overall operation has the effect of running $|P_A\rangle$ followed by $|P_B\rangle$, the program that is run is not $|P_A\rangle|P_B\rangle$. There is the intermediate step of replacing $|P_A\rangle$ with $|P_B\rangle$ and restarting execution, which, strictly speaking, are not program operations. In the context of $\mathcal{UQC}$ this could be achieved by initializing M with $|P_A\rangle$, running $|P_A\rangle$ and once the halt qubit is measured as $|1\rangle$, replacing the program portion of M with $|P_B\rangle$, setting the program counter register to 0 while leaving all other $\mathcal{UQC}$ registers and memory qubits unchanged, and clearing the halt qubit to resume execution of $|P_B\rangle$ with the results of the preceding program(s). This scheme, of course, works not only for branching programs but also for sequential programs.

Formally, our $\mathcal{UQC}$ program concatenation operator, $\Pi^{(n)}$, is defined as

$$\begin{aligned}\Pi^{(n)} \stackrel{\text{def}}{=} & \left(\sum_{i=1}^{n-1} |P_{i+1}\rangle\langle P_i|_{M(P)} \otimes |0\rangle\langle 1|_h \otimes \text{SWAP}_{P,P'(i)} \otimes |1\rangle\langle 0|_{S(i+1)}\right) \\ & + |P_1\rangle\langle 0|_{M(P)} \otimes |0\rangle\langle 0|_h \otimes |1\rangle\langle 0|_{S(1)} \\ & + \left(\sum_{i=1}^{n-1} |P_i\rangle\langle P_{i+1}|_{M(P)} \otimes |1\rangle\langle 0|_h \otimes |1\rangle\langle 0|_{S(i)}\right) \\ & + |P_n\rangle\langle P_n|_{M(P)} \otimes |1\rangle\langle 1|_h \otimes |1\rangle\langle 0|_{S(n)} \\ & + \sum_{i=1}^{n} |P_i\rangle\langle P_i|_{M(P)} \otimes |0\rangle\langle 1|_{S(i)} + \sum_{i=n+1}^{\infty} |P_i\rangle\langle P_i|_{M(P)},\end{aligned} \tag{3.28}$$

where n denotes the number of programs to be concatenated, $|P_i\rangle$ denotes the i-th program in the concatenation sequence (we are assuming that the programs have been enumerated such that the first n programs are the ones that we wish to concatenate), $M(P)$ denotes the program qubits portion of M, h denotes the halt qubit, P denotes the program counter register, $P'(i)$ denotes the i-th ancillary program counter register, and $S(i)$ is the i-th flag denoting that program i has been swapped. Note that some suitable finite unused subset of M can be used for $P'(i)$ and $S(i)$ since these are initialized to 0 and are only used once in the program concatenation operation. The P'

and S' arrays of qubits are required to save intermediate states during program swaps to ensure unitarity.

In order to concatenate n given programs, then, we simply modify the overall $\mathcal{UQC}$ evolution operator to

$$U \stackrel{\text{def}}{=} \Pi^{(n)} U_{\text{CF}} U_{\text{UP}} U_{\text{RI}} U_{\text{UF}} U_{\text{UD}} U_{\text{EX}} U_{\text{XD}} U_{\text{IF}} \tag{3.29}$$

U then has the net effect of running each program until it halts, swapping each completed program with the next program in the concatenation sequence, swapping the program counter register with 0, flipping the halt qubit (and hence starting execution of the swapped program), and leaving the final result on the tape when the last program, $|P_n\rangle$, halts. Even if the individual programs are known to halt, the concatenated program will not necessarily halt because, in general, the input data to the individual programs will change when run as part of a concatenated program. Hence, whether or not a concatenated program will halt is independent of whether or not its constituent programs halt.

The famous Turing halting problem is only relevant in the context of executing programs that can branch. Non-branching finite programs, by construction, will always halt so the halting problem is a moot point in that case. This raises a question about Deutsch's UQTM. Deutsch did not explicitly consider branching in his original UQTM proposal and thus it is unclear whether or not his program concatenation scheme rested on the assumption that UQTM programs were non-branching. If UQTM programs could involve branching, then without guaranteed halting of the concatenated programs, the validity of Deutsch's program concatenation scheme is problematic. Deutsch's description of his program concatenation scheme suggests that it was an 'appending' scheme rather than a 'swapping' scheme as we have defined.

There is still one problem with the program concatenation scheme. As currently defined, the halt qubit will be flipped several times during the course of executing a concatenation of programs (assuming that each constituent program halts, of course). Thus, it may appear that there is no way for an external observer (or classical computer) to distinguish between the intermediate and final states of the halt qubit. However, this does not pose a problem so long as the measurement of the halt qubit does not affect the result that we will ultimately measure on the memory tape in which case we simply measure the halt qubit, wait the time associated with program swap operations

(i.e. $|P_{i+1}\rangle\langle P_i|$) to be completed, and measure the halt qubit again. If the halt qubit was in an intermediate set state, we will then find it cleared. If, on the other hand, the halt qubit was in its final set state, then we will find it still set and we can then measure the memory tape to find the result. Thus, periodic measurements of the halt qubit suffice to identify whether the concatenated program has halted. The question, then, is whether periodic measurements of the halt qubit affect the final measurement of the memory tape. Ozawa [33] has already proven that periodic measurement of the halt qubit does not spoil the result of the computation (i.e. the final measurement of the memory tape contents). That is, the probability of finding the memory tape in state M_i after N iterations of U with periodic measurements (monitoring) of the halt qubit and the probability of finding the memory tape in the state M_i after N iterations of U without periodic measurements of the halt qubit (i.e. one single measurement of M after N iterations of U) are identical. Thus, periodic measurements of the halt qubit do not spoil the intermediate computation as Myers argued.

Therefore, we see that concatenation of $\mathcal{UQC}$ programs works in the same way as concatenation of classical computer programs. While the halting question for the resultant program still remains just as it does for classical computers, a valid unitary $\mathcal{UQC}$ program concatenation scheme exists. The programs to be concatenated are sequentially executed without changing the state of internal registers except for the program counter. Not surprisingly, the concatenation scheme is analogous to the classical case.

3.7 UQC and the Church-Turing Thesis

One of the central foundation tenets of theoretical computer science is the Church-Turing Thesis [38; 39]. According to Turing [38],

> *Every 'function which would be regarded as computable' can be computed by the universal Turing machine.*

This is an empirical statement though it has thus far survived over 70 years of testing and analysis.

A computable function is essentially a program or algorithm that can be computed by a universal Turing machine in *finite* time. The most representative function that is not computable is the "halting problem."

The "halting problem" is the question of whether or not a given program, labeled x, halts on the input of x. Formally, the problem is defined as:

$$h(x) = \begin{cases} 1, & \text{if program } x \text{ halts on input } x \\ 0, & \text{if program } x \text{ does not halt on input } x. \end{cases}$$

This has been proven to be uncomputable by a universal Turing machine. That is, there exists no algorithmic means for computing the value of the halting function h for all values of x in finite time.

Many non-trivial and practical problems have been shown to be equivalent to $h(x)$. In other words, if $h(x)$ were computable so would many other important problems. So what does this have to do with Physics?

Consider the "halting observable" $\hat{h}$ [40] defined as

$$\hat{h} \stackrel{\text{def}}{=} \sum_{x=0}^{\infty} h(x)|x\rangle\langle x|, \tag{3.30}$$

where $|x\rangle$ is an orthonormal basis for the state space of some physical system with a countably infinite dimensional state space. For example, the states $|x\rangle$ might be the number of states of a single mode of the EM field.

$\hat{h}$ is a valid quantum mechanical observable as it is Hermitian. In light of the Church-Turing Thesis, then, in principle, we must have either:

(1) It is possible to construct a device that can measure the observable $\hat{h}$,

(2) It is not possible to construct a device that can measure the observable $\hat{h}$.

If (1) were true, this would violate the Church-Turing Thesis, since one could prepare the system in the state $|x\rangle$ and measurement would yield $h(x)$ with probability 1. If one accepts the Church-Turing Thesis, then, (2) must be true. The analysis in [41] suggests that (2) must be true.

Namely, only a limited class of observables correspond to physically realizable measurements. It should be noted that there are ample examples of non-computable functions other than $h(x)$ so the limitation on observables is not necessarily trivial and inconsequential. Similarly, approximate measurements of $\hat{h}$, say $\hat{h}'$ would allow arbitrary approximations of $\hat{h}$ so they cannot be realizable either.

This implies that the class of physically realizable (measurable) observables is restricted to those that correspond to computable functions! Computer science would impose restrictions on physical reality.

How about dynamic evolutions? Consider a function g [40] that is defined as follows:

$$g(x) = \begin{cases} 2m-2, & \text{if } x \text{ is the } m\text{-th smallest non-negative integer such that } h(x)=0, \\ 2m-1, & \text{if } x \text{ is the } m\text{-th smallest non-negative integer such that } h(x)=1. \end{cases}$$

The operator

$$U \stackrel{\text{def}}{=} \sum_{x=0}^{\infty} |g(x)\rangle\langle x|,$$

is unitary.

$$U^\dagger U = \left(\sum_{x=0}^{\infty} |g(x)\rangle\langle x|\right)^* \left(\sum_{y=0}^{\infty} |g(y)\rangle\langle y|\right) = \sum_{x,y} |x\rangle\langle g(x)|g(y)\rangle\langle y| = \sum_{x} |x\rangle\langle x| = \mathbb{I}$$

Suppose that we prepare a system in the state $|x\rangle$, let it evolve according to U, and then perform a measurement in the $|x\rangle$ basis with the result x'. Then x' is even iff $h(x) = 0$ and x' is odd iff $h(x) = 1$ so this provides a means to compute the halting function. Once again, then, one of the following must be true in principle:

(1) It is possible to construct a system whose dynamics are described by U,

(2) It is not possible to construct a system whose dynamics are described by U.

Again, if we accept the Church-Turing Thesis, it places a restriction on the class of physically realizable Hamiltonians.

The existence of an explicit construction of a universal quantum computer [23] enables us to address this question on a concrete quantum mechanical footing. That is, we can recast the halting problem as a quantum mechanical problem and investigate whether the halting problem applies to quantum computers.

3.8 The Halting Problem

Let us recast halting problem in the context of $\mathcal{UQC}$'s time evolution as governed by quantum mechanics. The halting problem in our context can be stated as follows: For a given initial state (i.e. for a given program and initial data), does $\mathcal{UQC}$ ever reach a state where the halt qubit is in the $|1\rangle$ state after applying U, the time evolution

operator of $\mathcal{UQC}$, a **finite** number of times (i.e. in finite time)? We are interested in determining, for a given initial state, whether a measurement of the halt qubit will ever find it in the $|1\rangle$ state.

First of all, recall Ehrenfest's Theorem (see [42], for example):

$$\frac{d}{dt}\langle Q\rangle = \langle\frac{\partial}{\partial t}\hat{Q}\rangle + \frac{i}{\hbar}\langle[H,\hat{Q}]\rangle, \tag{3.31}$$

where $\langle Q\rangle$ is the expectation value of some observable $\hat{Q}$ and H is the Hamiltonian of the system. Now, consider the halt qubit observable, specifically the spectral projection of the halt qubit corresponding to the halted state:

$$\hat{O}_h = |1\rangle\langle 1|_h \otimes \mathbb{1}_{\neq h}. \tag{3.32}$$

Because the halt qubit observable $\hat{O}_h$ is time-independent, applying Ehrenfest's Theorem we get

$$\frac{d}{dt}\langle O_h\rangle = \frac{i}{\hbar}\langle[H,\hat{O}_h]\rangle, \tag{3.33}$$

where H denotes the Hamiltonian that generates U. We can immediately conclude that $\hat{O}_h$ must be incompatible with H because $\mathcal{UQC}$ programs always start with the halt qubit in the $|0\rangle$ state and if $\hat{O}_h$ and H were compatible, no program could ever halt. Stated differently $\frac{d}{dt}\langle O_h\rangle = 0$ for non-halting programs. Conversely, $\frac{d}{dt}\langle O_h\rangle \neq 0$ implies that a measurement of the halt qubit *can* find the machine in a halted state and thus that the program is a halting program. It should be noted that this does not imply anything about how likely it would be to find the machine and hence the program in a halted state. The probability may be small but, for our purposes, we are only interested in whether the program *can* halt in **finite** time.

First, we note that $[\hat{O}_h, U_{\neq \mathrm{EX}}] = 0$ but that $[\hat{O}_h, U] \neq 0$ because using equation (3.7) we find that

$$[\hat{O}_h, U_{\mathrm{EX}}] = |\mathrm{h}\rightarrow 1\rangle\langle \mathrm{h}\rightarrow 1|_I \otimes (|1\rangle\langle 0| - |0\rangle\langle 1|)_h. \tag{3.34}$$

It is also readily verified using equations (3.5) - (3.7) that

$$[\hat{O}_h, U_{\mathrm{EX}}^\dagger] = [\hat{O}_h, U_{\mathrm{EX}}], \tag{3.35}$$

$$[U_{\mathrm{XD}}^\dagger, [\hat{O}_h, U_{\mathrm{EX}}^\dagger]] = 0, \tag{3.36}$$

and that

$$U_{\mathrm{IF}}^{\dagger}[\hat{O}_h, U_{\mathrm{EX}}^{\dagger}]U_{\mathrm{IF}} = \delta_{M(P),|\mathrm{h}\rightarrow 1\rangle}(|1\rangle\langle 0| - |0\rangle\langle 1|)_h. \tag{3.37}$$

We thus obtain

$$\begin{aligned}
U^{\dagger}\hat{O}_h &= U_{\mathrm{IF}}^{\dagger}U_{\mathrm{XD}}^{\dagger}U_{\mathrm{EX}}^{\dagger}U_{\mathrm{UD}}^{\dagger}...U_{\mathrm{CF}}^{\dagger}\hat{O}_h && (3.38)\\
&= U_{\mathrm{IF}}^{\dagger}U_{\mathrm{XD}}^{\dagger}\left(\hat{O}_h U_{\mathrm{EX}}^{\dagger} - [\hat{O}_h, U_{\mathrm{EX}}^{\dagger}]\right)U_{\mathrm{UD}}^{\dagger}...U_{\mathrm{CF}}^{\dagger} && (3.39)\\
&= \hat{O}_h U^{\dagger} - U_{\mathrm{IF}}^{\dagger}[\hat{O}_h, U_{\mathrm{EX}}^{\dagger}]U_{\mathrm{XD}}^{\dagger}U_{\mathrm{UD}}^{\dagger}...U_{\mathrm{CF}}^{\dagger} && (3.40)\\
&= \hat{O}_h U^{\dagger} - U_{\mathrm{IF}}^{\dagger}[\hat{O}_h, U_{\mathrm{EX}}^{\dagger}]U_{\mathrm{IF}}U_{\mathrm{IF}}^{\dagger}U_{\mathrm{XD}}^{\dagger}U_{\mathrm{UD}}^{\dagger}...U_{\mathrm{CF}}^{\dagger} && (3.41)\\
&= \hat{O}_h U^{\dagger} + \delta_{M(P),|\mathrm{h}\rightarrow 1\rangle}(|0\rangle\langle 1| - |1\rangle\langle 0|)_h)U^{-\dagger}, && (3.42)
\end{aligned}$$

where $U^{-\dagger} = U_{\mathrm{IF}}^{\dagger}U_{\mathrm{XD}}^{\dagger}U_{\mathrm{UD}}^{\dagger}...U_{\mathrm{CF}}^{\dagger}$. Note that U^{-} is a unitary operator that has no effect on the halt qubit since the only component of $U^{\dagger}$ that affects the halt qubit is $U_{\mathrm{EX}}^{\dagger}$ which is absent from $U^{-\dagger}$. This yields

$$[U^{\dagger}, \hat{O}_h] = \delta_{M(P),|\mathrm{h}\rightarrow 1\rangle}(|0\rangle\langle 1| - |1\rangle\langle 0|)_h U^{-\dagger}, \tag{3.43}$$

which combined with equation (3.33) and the fact that $H = i\hbar(\partial_t U)U^{\dagger}$ yields

$$\begin{aligned}
\tfrac{d}{dt}\langle O_h\rangle = \langle &\tfrac{i}{\hbar}[HU, \hat{O}_h]U^{\dagger}\\
&+(\partial_t U)\delta_{M(P),|\mathrm{h}\rightarrow 1\rangle}(|0\rangle\langle 1| - |1\rangle\langle 0|)_h U^{-\dagger}\rangle.
\end{aligned} \tag{3.44}$$

Recognizing that $(|0\rangle\langle 1| - |1\rangle\langle 0|)_h = i(\sigma_y)_h$ and that $(\partial_t U) = \frac{1}{i\hbar}HU$, this can be expressed as

$$\frac{d}{dt}\langle O_h\rangle = \frac{1}{\hbar}\langle i[HU, \hat{O}_h]U^{\dagger} + \delta_{M(P),|\mathrm{h}\rightarrow 1\rangle}HU(\sigma_y)_h U^{-\dagger}\rangle. \tag{3.45}$$

Noting that $[(\sigma_y)_h, U^{-\dagger}] = 0$ since $U^{-\dagger}$ does not interact with the halt qubit, we get the general result

$$\frac{d}{dt}\langle O_h\rangle = \frac{1}{\hbar}\langle i[HU, \hat{O}_h]U^{\dagger} + HUU^{-\dagger}(\sigma_y)_h\rangle. \tag{3.46}$$

We now consider the cases where (3.46) is applied to any given program and initial data state $|\psi\rangle_{\neq h}|0\rangle_h \stackrel{\mathrm{def}}{=} |\psi, 0\rangle$. To simplify the derivation and without loss of generality, we will change the instruction encoding of |H⟩ from $|0100\rangle$ to $|1010\rangle$ (a previously unused instruction in Table 4.1), make $|0100\rangle$ an unused instruction encoding, and require that all programs have $|0100\rangle$ as the first instruction. This effectively causes U_{EX} to act as $\mathbb{1}$ when it encounters $|0100\rangle$ as can be readily verified from (3.7). Hence requiring $|0100\rangle$

as the first instruction in all programs has no effect on program execution and results. Applying equation (3.46) to an arbitrary program and initial data state we get

$$\frac{d}{dt}\langle O_h\rangle_\psi = \frac{i}{\hbar}\langle\psi,0|[HU,\hat{O}_h]U^\dagger|\psi,0\rangle + \frac{1}{\hbar}\langle\psi,0|HUU^{-\dagger}(\sigma_y)_h|\psi,0\rangle, \tag{3.47}$$

and it can be readily verified using equations (3.5)-(3.12) that $U^\dagger|\psi,0\rangle = |\psi',0\rangle$, so that

$$\frac{i}{\hbar}\langle\psi,0|[HU,\hat{O}_h]U^\dagger|\psi,0\rangle = \frac{i}{\hbar}\langle\psi,0|[HU,\hat{O}_h]|\psi',0\rangle \tag{3.48}$$

but because $\hat{O}_h|\psi,0\rangle = 0$, for all ψ, this means that

$$\frac{d}{dt}\langle O_h\rangle_\psi = \frac{1}{\hbar}\langle\psi,0|HUU^{-\dagger}(\sigma_y)_h|\psi,0\rangle, \tag{3.49}$$

and applying $(\sigma_y)_h$ to $|\psi,0\rangle$, we obtain

$$\frac{d}{dt}\langle O_h\rangle_\psi = \frac{i}{\hbar}\langle\psi,0|HUU^{-\dagger}|\psi,1\rangle. \tag{3.50}$$

We now expand $|\psi,1\rangle$ into the individual components of $\mathcal{UQC}$ to make it easier to follow the evolution as $UU^{-\dagger}$ is applied to $|\psi,1\rangle$:

$$|\psi,1\rangle = |h,x,D,P,F,H,s,I,M\rangle = |1,0,0,0,0,-1,0,0,0100...\rangle.$$

Using equations (3.5)-(3.12), we readily confirm that

$$\begin{aligned}
&U_{\mathrm{CF}}^\dagger|1,0,0,0,0,-1,0,0,0100...\rangle = |1,0,0,0,\mathrm{NEXT},1,0,0,0000...\rangle,\\
&U_{\mathrm{UP}}^\dagger|1,0,0,0,\mathrm{NEXT},1,0,0,0000...\rangle = |1,0,0,-5,\mathrm{NEXT},1,0,0,0000...\rangle,\\
&U_{\mathrm{RI}}^\dagger|1,0,0,-5,\mathrm{NEXT},1,0,0,0000...\rangle = |1,0,0,-5,\mathrm{NEXT},1,0,0,0000...\rangle,\\
&U_{\mathrm{UF}}^\dagger|1,0,0,-5,\mathrm{NEXT},1,0,0,0000...\rangle = |1,0,0,-5,\mathrm{NEXT},1,0,0,0000...\rangle,\\
&U_{\mathrm{UD}}^\dagger|1,0,0,-5,\mathrm{NEXT},1,0,0,0000...\rangle = |1,0,0,-5,\mathrm{NEXT},1,0,0,0000...\rangle,\\
&U_{\mathrm{XD}}^\dagger|1,0,0,-5,\mathrm{NEXT},1,0,0,0000...\rangle = |1,0,0,-5,\mathrm{NEXT},1,0,0,0000...\rangle,\\
&U_{\mathrm{IF}}^\dagger|1,0,0,-5,\mathrm{NEXT},1,0,0,0000...\rangle = |1,0,0,-5,\mathrm{NEXT},1,0,0,0000...\rangle,\\
&U|1,0,0,-5,\mathrm{NEXT},1,0,0,0000...\rangle = |1,0,0,0,0,-1,0,0,0100...\rangle = |\psi,1\rangle.
\end{aligned}$$

Thus we obtain the final result:

$$\frac{d}{dt}\langle O_h\rangle_\psi = \frac{i}{\hbar}\langle\psi,0|H|\psi,1\rangle. \tag{3.51}$$

Now, we can expand $|\psi,0\rangle$ and $|\psi,1\rangle$ in terms of the energy eigenstates of the machine as follows.

$$|\psi,0\rangle = \sum_i \alpha_i|\phi_i\rangle, \tag{3.52}$$

$$|\psi,1\rangle = \sum_j \beta_j|\phi_j\rangle, \tag{3.53}$$

where $|\phi_n\rangle$ denotes the n-th energy eigenstate of the machine. Then

$$\langle \psi, 0|H|\psi, 1\rangle = \sum_{i,j} \alpha_i^* \beta_j E_j \langle \phi_i | \phi_j \rangle = \sum_i \alpha_i^* \beta_i E_i,$$

and we obtain the final result

$$\frac{d}{dt}\langle O_h \rangle_\psi = \frac{i}{\hbar} \sum_i \alpha_i^* \beta_i E_i, \tag{3.54}$$

where the sum is over the energy eigenstates of the starting state of the machine.

We see that $\frac{d}{dt}\langle O_h \rangle$ is a function of the energy of the machine for the given program and initial data. Conceptually, this is to be expected. While a computation is in progress, $\mathcal{UQC}$ is a closed quantum system. Therefore, its energy is conserved. Moreover, because H is time-independent, the states of $\mathcal{UQC}$ are stationary states. As such, the set of energy eigenstates that comprise the initial starting state of the machine does not change over time. Stated differently, the machine's evolution is restricted to those states of the machine that are a linear combination of the initial energy eigenstates. Therefore, whether a program will halt is ultimately governed by the energy eigenstates of the initial state of the machine and whether those eigenstates can constitute a halted state.

Now, if we accept the Church-Turing Thesis then either one or all of α_i, β_i, and E_i cannot be determined. The question then is what prevents this? In principle, we can determine these if we can determine H from U. Yet, there must be an inherent problem that prevents this for the Church-Turing Thesis to hold. Unfortunately, this analysis is beyond the scope of this thesis and will require further investigation. What is important, however, is that the physical basis for the halting problem and hence the Church-Turing Thesis must be apparent in the Hamiltonian of the $\mathcal{UQC}$.

3.9 Discussion

The quantum computer we have defined is universal in the sense that, under the control of quantum programs, it can firstly emulate any classical Turing machine by being able to compute the NAND function and secondly can approximate any unitary operation to any desired accuracy by being able to apply the set of $\{\mathrm{H}, \mathrm{CNOT}, \mathrm{T}\}$ operations on a specified set of qubits. The machine also supports conditional branching and

hence conditional execution, a feature that is not directly possible in the quantum gate array circuit framework. The defined halting scheme works in a way that prevents changes to the memory tape once the program has halted thus satisfying Ozawa's proof requirement and allowing for a valid program concatenation scheme. Because of its universality, $\mathcal{UQC}$ serves as a prototypical model for general-purpose programmable quantum computation and should find uses in the development and analysis of quantum algorithms and complexity. The $\mathcal{UQC}$ should also find applications in the analysis of fundamental quantum computing questions such as the exploration of the physical basis of the halting problem that we briefly explored in this chapter.

4

Oracle Based Algorithms On A Universal Quantum Computer

4.1 Introduction

Quantum networks which connect quantum systems and can transmit quantum information have been extensively discussed [26]. Quantum connectivity provides a means of overcoming size-scaling and error-correction problems, and has significant advantages over classical connectivity. Furthermore, networks of quantum computers have also been proposed [27] where information can be exchanged between nodes via quantum and classical channels. A general question arises as to how such quantum computers can communicate and exchange information. In the simplest case a quantum computer may download data sets from other nodes over the quantum network, but in more complex cases use the network to call subroutines, or concatenate programs from other quantum computers.

It is well known that classical principles do not necessarily apply in the realm of quantum mechanics. The no-cloning theorem (see [43] for example) is a well-known example of this as is the ability to halt a programmable quantum computer as discussed in the previous chapter. Thus, it is imperative to formally show whether a classical solution or property is applicable (or even relevant) in the realm of quantum mechanics. Assuming that a classical solution to a problem directly applies to a quantum mechanical system is prone to run into potential complications.

We address here the question of whether and how a universal quantum computer

can access an external oracle, which may be regarded as a "black box" quantum device, possibly over a quantum network but in any case as a separate and external quantum system to the universal quantum computer itself. In fact, the oracle may be a program running on a remote universal quantum computer. It should be noted that this is a different problem from that of implementing an oracle "program" on a universal quantum computer. This is of course possible by virtue of the fact that the computer is universal. Hence, if a program exists, it can be implemented and executed on a universal quantum computer. The ability to utilize external quantum devices over a network connection, however, is a different problem because such devices are external to the universal quantum computer itself.

Classically, the ability to access devices on a network is a well-known problem with well-known solutions. However, as stated earlier, we cannot assume that this is necessarily the case for a quantum computer accessing quantum devices on a quantum network. Our aim is to explicitly show that accessing external quantum devices with a universal quantum computer is indeed possible by devising universal quantum computer programs to implement well-known oracle based quantum algorithms, namely the Deutsch, Deutsch-Jozsa, and the Grover algorithms using external black-box quantum oracle devices.

In the previous chapter we constructed a programmable universal quantum computer $\mathcal{UQC}$ that is universal in the sense that it can emulate any classical Turing machine and can approximate any unitary operation to any desired accuracy. It is programmable in the sense that the machine's operations are specified using a sequence of instructions in the same way as for classical computers. $\mathcal{UQC}$ also supports conditional branching and hence conditional execution, a feature that is not directly possible in the quantum gate array circuit framework. Moreover, $\mathcal{UQC}$ uses a halting scheme that allows for valid program concatenation, thus resolving issues with the original Universal Quantum Turing Machine (UQTM) proposed by Deutsch [21].

In order to use information from a quantum network in $\mathcal{UQC}$ programs, we need to devise a means of enabling $\mathcal{UQC}$ programs to access such remote information and use that information for local computations. We assume that remote quantum nodes exist and treat them as black boxes without any assumptions as to their internal structure or operational details. Without loss of generality, we assume that such devices accept a finite number of input qubits and generate a finite number of output qubits. The input

and output qubits may be shared, which is the case if the remote device functions in such a way as to alter the input qubits based on its function. We also assume, without loss of generality, that quantum network nodes have an "enable" qubit, $|\text{en}\rangle$, that controls when access is to begin, in order to let the device know when the input data has been prepared and is valid. We further assume, without loss of generality, that the nodes of the network generate their output data in less time than the time associated with a single iteration of $\mathcal{UQC}$. If the query time were longer than a single iteration of $\mathcal{UQC}$ or were data-dependent, one could simply write the $\mathcal{UQC}$ program to wait for the appropriate number of cycles before using the result of the network access. Alternatively, the nodes could provide an "access completed" status flag qubit such that the $\mathcal{UQC}$ program could poll this status flag qubit before using the result of a network access.

4.2 Accessing Networked Quantum Resources With $\mathcal{UQC}$

Recall from the previous chapter that $\mathcal{UQC}$ consists of a memory tape M with an infinite number of qubits, of which only a finite portion is ever used, and a processor that contains observables that play the roles of several registers, including a data register D, a program counter register P, a scratch qubit s, and the halt qubit h. The processor executes programs stored on the memory tape using data that is also stored on the memory tape. A program of $\mathcal{UQC}$ consists of a sequence of qubits whose states encode instructions of the instruction set defined in the previous chapter and reproduced in Table 4.1 below.

The single qubit operations H and T act on the qubit at tape location $M(D)$, denoted $|M\rangle_D$, and the two qubit operations SWAP and NAND act on $|M\rangle_D$ and the scratch qubit $|s\rangle$, the latter being used as the control qubit for the NAND operation.

The instruction set includes a set of operations that can approximate any unitary operation to any desired accuracy. Thus, it is quantum computationally universal. In the previous chapter we constructed a $\mathcal{UQC}$ program that can compute the NAND function, thereby showing that the machine can compute any classically computable function. Because of $\mathcal{UQC}$'s universality, any algorithm that can be implemented in the quantum gate array framework can be mapped to an equivalent $\mathcal{UQC}$ program by virtue of the fact that gate array circuits can be decomposed into circuits of gates with

Table 4.1: $\mathcal{UQC}$ Instruction Set

Label	Encoding	Description
$\|\text{NOP}\rangle$	$\|0000\rangle$	No operation
$\|D \rightarrow 0\rangle$	$\|0001\rangle$	$D \rightarrow 0$
$\|D + 1\rangle$	$\|0010\rangle$	$D \rightarrow D + 1$
$\|D - 1\rangle$	$\|0011\rangle$	$D \rightarrow D - 1$
$\|\text{H}\rangle$	$\|0100\rangle$	Apply Hadamard operation to $\|M\rangle_D$
$\|\text{T}\rangle$	$\|0101\rangle$	Apply $\pi/8$ operation to $\|M\rangle_D$
$\|\text{SWAP}\rangle$	$\|0110\rangle$	$\|M\rangle_D \leftrightarrow \|s\rangle$
$\|\text{CNOT}\rangle$	$\|0111\rangle$	CNOT of $\|M\rangle_D$ and $\|s\rangle$ ($\|s\rangle$: control)
$\|D \leftrightarrow P\rangle$	$\|1000\rangle$	$\|D\rangle \leftrightarrow \|P\rangle$ (branch) iff $s = 0$
$\|\text{CLS}\rangle$	$\|1001\rangle$	Clear s
$\|h \rightarrow 1\rangle$	$\|1111\rangle$	$\|h\rangle \rightarrow \|1\rangle$ (set halt qubit)

the same universal set of unitary operations $\{\text{H}, \text{T}, \text{CNOT}\}$ that are implemented in the $\mathcal{UQC}$ instruction set. Each of the qubits in a quantum circuit (i.e. lines connecting gates) can be mapped to a suitable memory tape data qubit and each of the unitary operations (i.e. quantum gates) can be mapped to a suitable $\mathcal{UQC}$ subroutine. It is possible therefore to map quantum gate array implementations of algorithms such as the quantum Fourier transform, quantum phase estimation, quantum order-finding, quantum factoring discussed in [1] (Chapter 5) onto $\mathcal{UQC}$.

Modifying $\mathcal{UQC}$ to use networked quantum devices, then, is a matter of connecting the qubits comprising the interface (input, output, enable, and optionally access complete) qubits of those devices to a finite subset of the data portion of M, which is the quantum analog of a classical computer's memory-mapped I/O and allows $\mathcal{UQC}$ programs to access remote devices using the M qubits that are connected to those devices. The $\mathcal{UQC}$ programs prepare the appropriate input data qubits, set the corresponding access enable qubits to perform an access, and utilize the corresponding output data qubits of M. It should be noted that a remote quantum device could be another instance of $\mathcal{UQC}$ which would enable distributed quantum computing. However, the scheme to access data from remote devices, be they simple devices or full-fledged quantum computers, would work in the same way.

4.3 Primitive Programs

Using the primitive programs defined in the previous chapter, we define $|P_i^{\mathrm{X}}\rangle$ as the program that applies the σ_x (X) operation on data qubit $i \in \mathbb{Z}^+$. Noting that a CNOT operation with the control qubit in the $|1\rangle$ state is equivalent to the X operation, we deduce the equivalence

$$|P_i^{\mathrm{X}}\rangle \equiv |P_{|1\rangle,i}^{\mathrm{C}}\rangle, \tag{4.1}$$

where the subscript $|1\rangle$ denotes that some suitable data qubit on the memory tape has been prepared in the state $|1\rangle$. Similarly, we define $|P_i^{\mathrm{Z}}\rangle$ as the program that applies the σ_z (Z) operation on data qubit $i \in \mathbb{Z}^+$. Noting that HXH = Z, we deduce

$$|P_i^{\mathrm{Z}}\rangle \equiv |P_i^{\mathrm{H}}\rangle|P_i^{\mathrm{X}}\rangle|P_i^{\mathrm{H}}\rangle. \tag{4.2}$$

Finally, we define a program $|P_{i,j}^{\mathrm{CZ}}\rangle$ that conditionally applies the Z operation on data qubit $i \in \mathbb{Z}^+$ and data qubit $j \in \mathbb{Z}^+$. Since CNOT is the conditional X operation, we have

$$|P_{i,j}^{\mathrm{CZ}}\rangle \equiv |P_j^{\mathrm{H}}\rangle|P_{i,j}^{\mathrm{C}}\rangle|P_j^{\mathrm{H}}\rangle. \tag{4.3}$$

4.4 $\mathcal{UQC}$ Algorithms Using Networked Quantum Oracle Devices

With the notable exception of Shor's factorization algorithm [2], several well known quantum algorithms that achieve a speed-up over their fastest known classical counterparts rely on the use of an oracle, the best known examples being the Deutsch, Deutsch-Jozsa, and Grover algorithms (see Nielsen and Chuang [1], for example). The Deutsch algorithm can determine a global property of a function $f(x)$, namely $f(0) \oplus f(1)$, using only one evaluation of $f(x)$ whereas the fastest classical algorithm requires at least two evaluations of $f(x)$. The Deutsch-Jozsa algorithm can determine whether a two-valued (0 or 1) function $f(x)$ is constant or balanced with only one evaluation of $f(x)$ whereas the fastest classical algorithm requires $2^{n-1}+1$ evaluations, where n denotes the number of bits required to encode the possible values of $f(x)$. Grover's algorithm [5] can find a marked item in an unstructured database of N elements in $O(\sqrt{N})$ operations whereas the fastest classical algorithm requires $O(N)$ operations. Thus, these quantum algorithms all achieve at least a quadratic speedup over their classical counterparts.

These algorithms are well suited to illustrate the use of networked quantum resources with the $\mathcal{UQC}$ because they rely on black-box quantum devices that generate some output based on the given input. They thus serve as prototypical examples of a networked quantum node, whose internal implementation details are unknown; only the interface protocol need be known. Here, we assume the simplest protocol, which is that the output is valid one "clock cycle" after making a request.

4.4.1 Deutsch and Deutsch-Jozsa Algorithms on $\mathcal{UQC}$

We now illustrate the use of a networked quantum device in a $\mathcal{UQC}$ program by first implementing the simplest known oracle based quantum algorithm, Deutsch's algorithm. The Deutsch oracle works as follows:

$$|x, y\rangle \rightarrow \begin{cases} |x, y \oplus f(x)\rangle & \text{if } |\text{en}\rangle = |1\rangle, \\ |x, y\rangle & \text{otherwise,} \end{cases}$$

where f is some function and $|\text{en}\rangle$ denotes the oracle query enable flag. The memory tape is prepared with $D(0) = |0\rangle$ and $D(1) = |1\rangle$ where $D(0)$ and $D(1)$ take the roles of x and y, respectively. We assume without loss of generality that $D(2)$ takes the role of $|\text{en}\rangle$ and is prepared as $|0\rangle$, and $D(3)$ is initially prepared as $|1\rangle$.

The program that executes the Deutsch algorithm is

$$|P_D\rangle \stackrel{\text{def}}{=} |P^{\text{H}}_{1,0}\rangle|S_{2,3}\rangle|S_{2,3}\rangle|P^{\text{H}}_{0}\rangle|P_h\rangle, \tag{4.4}$$

where $|P^{\text{H}}_{1,0}\rangle$ applies the Hadamard transform to the data qubits corresponding to x and y. $|S_{2,3}\rangle|S_{2,3}\rangle$ swap qubits $D(2)$ and $D(3)$ thereby setting the oracle's $|\text{en}\rangle$ qubit (recall that $D(2)$ is connected to $|\text{en}\rangle$ and that $D(2) = |0\rangle$ and $D(3) = |1\rangle$ initially) for a single $\mathcal{UQC}$ cycle and then clears it, returning the state of $D(3:2)$ back to the original state. At this point, the oracle has generated the output state $|D(0), D(0) \oplus D(1)\rangle$. $|P^{\text{H}}_{0}\rangle$ then applies the Hadamard transform to the x output of the oracle and $|P_h\rangle$ halts the program thus yielding the following on the memory tape:

$$|D(0), D(1)\rangle = \pm|f(0) \oplus f(1)\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right].$$

Measuring $D(0)$ yields the result that we were interested in, $f(0) \oplus f(1)$. This is a specific mapping of the gate array implementation of the algorithm (see [1] Figure 1.19, for example) onto the instruction set of $\mathcal{UQC}$.

We can similarly implement the Deutsch-Jozsa algorithm by mapping a gate array implementation such as the one shown in [1], Figure 1.20. In this case, data qubits $D(0:n-1)$ take the role of x, $D(n)$ takes the role of y, and we use $D(n+1)$ as the $|\text{en}\rangle$ qubit. As before, $D(0:n-1)$ are prepared in the $|0\rangle$ state, $D(n)$ is prepared in the $|1\rangle$ state, $D(n+1)$ is prepared in the $|0\rangle$ state and $D(n+2)$ is prepared in the $|1\rangle$ state. The Deutsch-Jozsa oracle works like the Deutsch oracle with the only difference being that x is n qubits wide. The resulting $\mathcal{UQC}$ program that computes the Deutsch-Jozsa algorithm is therefore

$$|P_{DJ}\rangle \stackrel{\text{def}}{=} |P^{\text{H}}_{n-1,0}\rangle|S_{n+1,n+2}\rangle|S_{n+1,n+2}\rangle|P^{\text{H}}_{n-1,0}\rangle|P_h\rangle, \tag{4.5}$$

which is again a direct mapping of the gate array implementation onto the $\mathcal{UQC}$ instruction set.

4.4.2 Grover's Algorithm on $\mathcal{UQC}$

We now use the techniques developed in the previous section to implement the Grover unstructured database search algorithm. We assume that the database has only one marked solution as can be determined by using the quantum counting algorithm (see [1] Chapter 6, for example). We denote the query data qubits as $|q\rangle$ and the query enable flag as $|\text{en}\rangle$. The Grover oracle works as follows:

$$|q\rangle \rightarrow \begin{cases} (-1)^{f(q)}|q\rangle & \text{if } |\text{en}\rangle = |1\rangle, \\ |q\rangle & \text{otherwise} \end{cases}$$

where $f(q) = 1$ if q is a solution to the search problem and $f(q) = 0$ otherwise. More concisely, the oracle performs the unitary transformation

$$U_m \stackrel{\text{def}}{=} \mathbb{1} - 2|m\rangle\langle m|, \tag{4.6}$$

where $|m\rangle$ denotes the marked solution. In other words, the oracle flips the phase of the solution state but leaves non-solution states unchanged. Grover's algorithm prepares an initial query state as the equal superposition of all elements in the database, followed by $O(\sqrt{2^n})$ iterations of G, where

$$G \stackrel{\text{def}}{=} (2|s\rangle\langle s| - \mathbb{1})U_m, \tag{4.7}$$

and

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle \tag{4.8}$$

denotes the equal superposition of all database elements.

Thus, the first step in the program is to create a superposition of all database items in $D(n:1)$ where $D(i) = M(5i-1)$, $i \in \mathbb{Z}^+$, as the first query input. This is accomplished by the multiple qubit Hadamard primitive program $|P^{\mathrm{H}}_{n,1}\rangle$ defined in Eq. (3.21). The next step is to perform an oracle query. The following program performs an oracle call with query data prepared in $D(n:1)$:

$$|P_m\rangle \stackrel{\mathrm{def}}{=} |S_{n+1,n+2}\rangle |S_{n+1,n+2}\rangle, \tag{4.9}$$

where $D(n+1)$ is used as the oracle query enable qubit and $D(n+2)$ is initialized to $|1\rangle$. $D(n+1)$ is assumed to be initialized to $|0\rangle$ (i.e. the oracle query data is disabled at start-up). This program simply sets the query enable qubit for a single $\mathcal{UQC}$ cycle and then clears it, returning the state of $D(n+2:n+1)$ back to the original state. Thus, upon running $|P_m\rangle$, the result of the oracle call is in $D(n:1)$, i.e. this program is functionally equivalent to U_m.

The next step is to implement a program $|P_s\rangle$ that performs the reflection of a given state about the superposition of all basis states $|s\rangle$. This requires a conditional-phase operation that works as follows:

$$|x\rangle \rightarrow \begin{cases} |x\rangle & \text{if } x = 0, \\ -|x\rangle & \text{otherwise} \end{cases}$$

where $|x\rangle$ is n qubits wide. Up to a global phase, this can be implemented using the following procedure:

1. Apply the σ_x operation to all n qubits.
2. Apply a controlled-Z operation using $n-1$ qubits as control qubits and the remaining qubit as the data qubit.
3. Apply the σ_x operation to all n qubits.

We can construct a multiple qubit controlled-Z program $|P^{\mathrm{CZ}}_{i,j,k}\rangle$ where qubits i through j are the control qubits and qubit k is the data qubit, with the $|P^{\mathrm{CZ}}_{i,j}\rangle$ program defined in Eq. (4.3) and the Toffoli program $|P^{\mathrm{Toff}}_{i,j,k}\rangle$ that we defined in Chapter 5 using a procedure

analogous to that described in [1], Chapter 4. Armed with $|P^{\mathrm{CZ}}_{i,j,k}\rangle$, we construct $|P_s\rangle$ as follows:

$$|P_s\rangle \stackrel{\mathrm{def}}{=} |P^{\mathrm{H}}_{n,1}\rangle|P^{\mathrm{X}}_{n,1}\rangle|P^{\mathrm{CZ}}_{2,n,1}\rangle|P^{\mathrm{X}}_{n,1}\rangle|P^{\mathrm{H}}_{n,1}\rangle. \tag{4.10}$$

It can be readily verified that this is functionally equivalent to the $2|s\rangle\langle s| - \mathbb{1}$ operator. Thus, a program that performs a single Grover iteration is

$$|P_G\rangle \stackrel{\mathrm{def}}{=} |P_m\rangle|P_s\rangle. \tag{4.11}$$

In summary, the complete program to search a database of 2^n items with a single marked solution is

$$|\mathcal{G}\rangle \stackrel{\mathrm{def}}{=} |P^{\mathrm{H}}_{n,1}\rangle \left(|P_G\rangle\right)^{N_G} |P_h\rangle, \tag{4.12}$$

where $N_G = \left\lceil \frac{\pi}{4}\sqrt{2^n} \right\rceil$ is the number of Grover iterations that can be pre-computed based on the database size, or that $\mathcal{UQC}$ can compute from the database size using a classical algorithm. Upon execution of $|\mathcal{G}\rangle$, a measurement of $D(n:1)$ reveals the solution $|m\rangle$. Because there are no oracle queries associated with $|P^{\mathrm{H}}_{n,1}\rangle$ and $|P_h\rangle$, we immediately identify the complexity (as a measure of the number of oracle queries) of $|\mathcal{G}\rangle$ as N_G. Of course, this is identical to the number of oracle queries associated with an implementation in the gate array framework.

4.5 Discussion

We have presented a scheme to enable the universal quantum computer to utilize networked quantum resources. We have illustrated the scheme by describing $\mathcal{UQC}$ programs that implement the well-known oracle based Deutsch, Deutsch-Jozsa, and Grover algorithms using networked quantum oracle devices. We have therefore demonstrated how universal quantum computers can access networked quantum devices in a way analogous to that by which classical computers access network resources.

5

Symmetry Based Partial Search

5.1 Introduction

The Grover quantum search algorithm achieves a quadratic speedup over classical unstructured database search algorithms [5]. It has been shown that the Grover search algorithm is theoretically optimal assuming the use of the standard Grover oracle (see [29] for example). In some applications, however, a full search of an unstructured database is not necessary and a *partial* search suffices and would be expected to be faster because less information is required.

A partial search is practical when one does not need to find the exact item but those items that might have some property in common. For example, one might be interested in finding the people that live in a given suburb of some city. Intuitively, this is essentially the problem of searching a database of $\frac{N}{b}$ elements, where N is the number of elements in the database and b is the 'bin' or 'group' size. The bin size can also be considered as the 'fuzziness' or granularity of the search result. For the purpose of this paper, we will assume that the database of size N is divided into K blocks of equal size $b = \frac{N}{K}$. Without loss of generality, we may further assume that N and K are chosen such that $N = 2^n$, and $K = 2^k$, giving $b = 2^{n-k}$. The goal is to find an item that is in the same bin as the target item.

Several ways of achieving fast partial searches have been proposed [44] [45] [46] [47] [48] [49] [50]. The GRK algorithm presented in [51] is the fastest known partial search algorithm but there is no proof that this is optimal, see [48] for comparisons with other combinations of global and local iterations, and requires $O(\sqrt{N} - R\sqrt{b})$ oracle calls,

where R is a positive coefficient which may be maximized by a combination of local and global Grover iterations. By comparison, a full Grover search requires $O(\sqrt{N})$ oracle calls. If b is an order of magnitude smaller than N, the speedup of such a partial search compared to a full search is not significant. Moreover, according to [51], the lower bound for the number of oracle queries in any quantum partial search algorithm is $\frac{\pi}{4}\left(1-\frac{1}{\sqrt{K}}\right)\sqrt{N}$.

It is important to note that the optimal lower bounds associated with Grover search algorithms apply only to unitary search algorithms. That is, the optimal bounds are all derived based on the assumption that the algorithms proceed by performing a sequence of unitary operations essentially of the form $\prod_{i=1}^{k}(U_i G)$, where U_i is some unitary operator, G is the Grover oracle, and k is the number of iterations required to achieve a successful search. As such, these lower bounds do not necessarily apply if we were to replace U_i with non-unitary operators, such as projections, or replace G with different oracles, particularly non-unitary ones. Our hope, then, is to escape the lower bound by allowing for projections and assuming the use of the standard Grover oracle. In this chapter we explore a quantum partial search algorithm based on projections in an attempt to achieve a complexity of $O(\sqrt{\frac{N}{b}})$, which is the theoretical limit for a full Grover search of $\frac{N}{b}$ elements assuming the existence of an oracle which identifies the correct bin. This is conceptually the limit for an oracle with the property that it marks the bin with the target item.

In the process, we will devise a partial oracle using the standard Grover oracle that marks the bin containing the target item. This oracle will give rise to a symmetry based partial search algorithm but implementing the corresponding quantum circuit would require deterministically copying quantum states. If quantum states could be copied, the algorithm would lead to exponentially fast full searches, something that cannot be realized unless non-linear quantum processes are ever observed. Based on the general form of this oracle, we explore two realizable algorithms that involve the application of a symmetrization operator either at every iteration step or in the last iteration step. Both algorithms involve a well defined number of oracle calls but are not viable because the probabilities of measuring the target bin at the end turn out to be small in both cases.

5.2 Symmetry Based Partial Search

As stated earlier, we assume that the database of size N will be divided into K bins, each containing $b = \frac{N}{K}$ elements. In our case, however, the membership of bins will not be contiguous. Items $|0\rangle$, $|K\rangle$, $|2K\rangle$, ..., $|N-K\rangle$ will be in one bin. Items $|1\rangle$, $|K+1\rangle$, $|2K+1\rangle$, ..., $|N-K+1\rangle$ will be in the next bin, etc., with the last bin containing $|K-1\rangle$, $|2K-1\rangle$, ..., $|N-1\rangle$. Thus a given item $|x\rangle$ is in the same bin as all the items that have index $|x \mod K\rangle$.

When our search is executed, all items with index $|t \mod K\rangle$, where $|t\rangle$ is the target item, will be treated as equivalent targets. We will achieve this by creating a partial search oracle using the standard Grover oracle such that it marks (inverts the probability amplitude) all states $|t \mod K\rangle$, and not just $|t\rangle$. We avoid making any changes to the standard Grover oracle itself in order to avoid making any assumptions about the inner workings of the oracle. Moreover, this will enable us to characterize the complexity of our algorithm in terms of the number of standard Grover oracle calls thus allowing for a meaningful characterization of our algorithm's complexity. Our overall algorithm, then, is to run a global Grover search but using our partial oracle instead of the standard Grover oracle.

5.3 The Partial Search Oracle

We begin by defining a *shift operator* S_K which takes a state $|x\rangle$ and shifts it by K positions to the left. If $x < K$, then S_K wraps around to the corresponding state with index $\geq N-K$. That is, $S_K|x\rangle = |x-K \mod N\rangle$. For notational simplicity, we define $|x\rangle_y \stackrel{\text{def}}{=} |x \mod y\rangle$. Formally,

$$S_K = \sum_{i=0}^{N-1} |i\rangle_N\langle i+K|. \tag{5.1}$$

It is readily verified that S_K is unitary, that $S_K^\dagger = S_{-K}$, and that $S_{\ell K} = S_K^\ell$, where ℓ is some integer.

Therefore, to create an oracle that marks a state of the form $|t\rangle_K$, i.e. $|t+\ell K\rangle_N$, where $0 \leq \ell \leq b-1$, we shift the state to $|t\rangle$, we apply the standard Grover oracle, which flips it to $-|t\rangle$, and we then shift it back to $-|t+\ell K\rangle_N$. To confirm this we note that

the standard Grover oracle that marks the state $|t\rangle$ can be expressed as $O_t = \mathbb{1} - 2|t\rangle\langle t|$. Therefore, our shifted oracle is

$$O_{t+K} \stackrel{\text{def}}{=} S_K^\dagger O_t S_K = \mathbb{1} - 2|t+K\rangle_{N\ N}\langle t+K|.$$

We next modify our shifted oracle such that it marks all of the modulo K solutions at once. As in a standard Grover search, our starting state will be the equal superposition of all database items. A superposition is then formed over every shift of ℓK positions, with a shift to the left summed over values of ℓ. We thus have a sum over b states, where in each term of the summation, the state is shifted such that every modulo K solution is in the position of the target. Hence when the standard oracle is applied, each modulo solution is marked. We then undo the shifts with summed shifts to the right, restoring the modulo targets back to their original positions. In this way, we produce a state in which every modulo K target has been marked whereas the non-modulo K items remain unchanged. Formally, the un-normalized operator that achieves this is given by

$$\sum_{\ell=0}^{b-1} S_{\ell K}^\dagger O_t S_{\ell K}.$$

After this operator has been applied, each non-modulo K position is the sum of b of the initial states, none of which is the target, and thus has an amplitude $\sim b$. Each modulo K target state, however, is the sum of $b-1$ states which are not the target, and therefore has a positive amplitude, and one state which is the target, and therefore has equal but negative amplitude. Thus these states have amplitudes $\sim (b-2)$. This is not quite the desired state, however. We need to return the non-target positions to the amplitudes that they had before the oracle call. In essence they need to be 'untouched,' as in the standard Grover. We also need to drop the modulo K targets back to an equal but negative amplitude. To be precise, they will be flipped, relative to their initial values. Finally, we need to re-normalize the state. Thus, we define our unitary *partial search oracle* that accomplishes this as

$$O_{t_K} \stackrel{\text{def}}{=} (1-b)\mathbb{1} + \sum_{\ell=0}^{b-1} S_{\ell K}^\dagger O_t S_{\ell K}, \tag{5.2}$$

where t_K denotes $t \mod K$, signifying that this oracle marks all modulo K target items.

Now we will explicitly check that this operator does, in fact, have the desired effect, and that it is unitary. We have $S_{\ell K}^{\dagger}|t\rangle = |t + \ell K\rangle_N$, and so

$$S_{\ell K}^{\dagger} O_t S_{\ell K} = \mathbb{1} - |t + \ell K\rangle_N \, {}_N\langle t + \ell K|$$

for each $\ell = 0, \ldots b-1$. By summing over ℓ and using the definition (5.2) we obtain

$$O_{t_K} = \mathbb{1} - 2 \sum_{\ell=0}^{b-1} |t + \ell K\rangle_N \, {}_N\langle t + \ell K|. \tag{5.3}$$

The oracle O_{t_K} therefore marks any item of the form $|t + \ell K\rangle_N$, i.e. we have $O_{t_K}|x\rangle = -|x\rangle$ for all elements $|x\rangle$ in the target bin, and $O_{t_K}|x\rangle = |x\rangle$ for all other elements. O_{t_K} is also unitary and Hermitian, and so satisfies $(O_{t_K})^2 = \mathbb{1}$. We can clearly see that this is an oracle that marks any solution of the form $|t + \ell K\rangle$, that is to say, an oracle that marks all modulo K targets. It is also readily verified to be unitary. The idea is to substitute our oracle for the one in the standard Grover algorithm, such that after $\sim \frac{\pi}{4}\sqrt{\frac{N}{b}} = \frac{\pi}{4}\sqrt{K}$ iterations, we hope to achieve a high probability of measuring one of the modulo K solutions. Namely, our partial search iteration would be $G_K \stackrel{\text{def}}{=} A O_{t_K}$, where A is the inversion about the average operator, formally defined as

$$A = 2|\sigma\rangle\langle\sigma| - \mathbb{1}, \tag{5.4}$$

with $|\sigma\rangle$ denoting the superposition over all basis states.

Unfortunately, as is evident from (5.2) each call to O_{t_K} involves multiple calls to the standard Grover oracle O_t. This would defeat any savings and, in fact, would be less efficient than a standard full Grover search because the complexity of a partial search based on this oracle would have a lower bound of $O(\sqrt{bN})$ standard oracle calls. One way to reduce this overhead is restrict the partial search oracle to the subspace of symmetric states that will be defined in the following section.

5.4 Symmetric States

O_{t_K} can be optimized to eliminate the initial shift symmetrization at the expense of unitarity. This is possible because the starting state is the superposition of all basis states and hence applying the shift operator has no effect, i.e. $S_{\ell K}|\sigma\rangle = |\sigma\rangle$ for all ℓ. We have therefore

$$O_{t_K}|\sigma\rangle = \Big[(1-b)\mathbb{1} + bP_K O_t\Big]|\sigma\rangle, \tag{5.5}$$

where

$$P_K \stackrel{\text{def}}{=} \frac{1}{b}\sum_{\ell=0}^{b-1} S_{\ell K}^{\dagger}. \tag{5.6}$$

Since $S_K^{\dagger} = S_{-K}$ we have $P_K = P_K^{\dagger}$, and from $S_b K^{\dagger} = \mathbb{1}$ it follows that $P_K S_K^{\dagger} = P_K = S_K^{\dagger} P_K$. Hence $P_K S_{\ell K}^{\dagger} = P_K$ for all integers ℓ and therefore $P_K^2 = P_K$. We also have $S_K P_K = P_K S_K = P_K$.

It is convenient to change from the basis $\{|i\rangle\}$ to a basis consisting of the eigenstates $|\psi\rangle$ of the projection operator P_K. Since P_K and S_K commute we may simultaneously diagonalize P_K and S_K. S_K is unitary and therefore has complex eigenvalues of unit modulus, specifically $S_K|\psi\rangle = e^{i\theta}|\psi\rangle$ where $e^{i\theta}$ is a b-th root of unity, i.e. θ takes values $2\pi\ell/b$ for $\ell = 0, 1 \ldots b-1$. The eigenspace of S_K for each such ℓ has dimension K. The relation $P_K S_K = P_K$ implies $e^{i\theta} P_K|\psi\rangle = P_K|\psi\rangle$, so if $\theta \neq 0$ the corresponding eigenstate $|\psi\rangle$ lies in the nullspace of P_K. If $\theta = 0$ we have $S_K|\psi\rangle = |\psi\rangle$ and hence, from the definition (5.6), $P_K|\psi\rangle = |\psi\rangle$. The K eigenstates $|\psi\rangle$ of S_K with unit eigenvalue may be determined explicitly by expanding $|\psi\rangle$ in terms of the basis $\{|i\rangle\}$ and are given by

$$|s_\alpha\rangle \stackrel{\text{def}}{=} \frac{1}{\sqrt{b}}\sum_{\ell=0}^{b-1} |\alpha + \ell K\rangle_N, \tag{5.7}$$

for $\alpha = 0, 1, \ldots K-1$. Each state $|s_\alpha\rangle$ is the symmetric combination of all states in the bin α. Let us denote the linear span of the states $\{|s_\alpha\rangle\}$ by S, comprising the range of P_K, and denote its orthogonal complement by $S^{\perp}$, the $N-K$ dimensional nullspace of P_K. This space is spanned by the orthonormal set $\{|s_\beta^{\perp}\rangle\}$ for $\beta = 0, 1 \ldots N-K-1$, where $|s_\beta^{\perp}\rangle$ is an eigenstate of S_K corresponding to an eigenvalue not equal to unity, however we do not require the explicit form of these states. We refer to the elements $|s\rangle$ of S as symmetric states, and the elements $|s^{\perp}\rangle$ of $S^{\perp}$ as nonsymmetric states.

5.4.1 Construction of O_{t_K} Using Symmetric States

We note that

$$|\sigma\rangle = \frac{1}{\sqrt{K}}\sum_{\alpha=0}^{K-1} |s_\alpha\rangle, \tag{5.8}$$

hence $P_K|\sigma\rangle = |\sigma\rangle$ and therefore $P_K A = A + \mathbb{1} - P_K = A P_K$. For all states $|s\rangle$ in S, in which case $P_K|s\rangle = |s\rangle$, we have from (5.2)

$$O_{t_K}|s\rangle = (1-b)|s\rangle + b P_K O_t|s\rangle \tag{5.9}$$

and hence

$$G_K|s\rangle = (1-b)A|s\rangle + bP_K A\, O_t|s\rangle. \tag{5.10}$$

We do not expect O_{t_K} to be expressible in the form $P'O_t$, where P' is some projection operator because that would lead to a contradiction of the aforementioned established lower bounds for partial search algorithms based on the Grover oracle.

Since $P_K G_K|s\rangle = G_K|s\rangle$ we see that $G_K|s\rangle$ is a symmetric state provided that $|s\rangle$ is a symmetric state, i.e. if the domain of G_K is restricted to S then the range of G_K also lies in S. The state $A\, O_t|s\rangle$ is a superposition of symmetric and nonsymmetric states but the nonsymmetric components are projected out by P_K. Hence we may iterate G_K any number of times and the resultant states are all elements of S, provided only that the starting state lies in S. In effect G_K acts like a Grover operator, except that it acts on the K bins as represented by the states $|s_\alpha\rangle$, not on the individual database elements $|i\rangle$, and rotates the initial state to the target state with $n_K = \left\lceil \frac{\pi}{4}\sqrt{K} \right\rceil$ iterations.

The operation of O_{t_K} on $|\sigma\rangle$ according to (5.9) is depicted in Figure 5.1. The initial state $|\sigma\rangle$ is shown in a), with all N qubits having equal amplitude measured in units of $1/\sqrt{N}$, and $O_t|\sigma\rangle$ is shown in b) with a flipped target item. The state $bP_K O_t|\sigma\rangle$ is shown in c), where bP_K acts on b) according to the sum (5.6), shifting all elements by ℓK units to the left to create b terms which are summed over ℓ. The state $(1-b)|\sigma\rangle$ is shown in d) and is added to c) to produce $O_{t_K}|\sigma\rangle$ in e). Hence O_{t_K} creates a target bin and flips the sign of all items in that bin, leaving other items unchanged. This completes the construction of a partial search oracle using the standard Grover oracle.

5.4.2 Implementation of O_{t_K} and Discussion

At first glance, O_{t_K} appears to involve just a single standard oracle call. However, we will now show that such is not the case. Firstly, we define the operator

$$U_K \stackrel{\text{def}}{=} \mathbb{1} - 2P_K = \mathbb{1} - 2\sum_{\alpha=0}^{K-1} |s_\alpha\rangle\langle s_\alpha|,$$

which is both Hermitian and unitary and hence satisfies $U_K^2 = \mathbb{1}$. This operator leaves nonsymmetric states unchanged and satisfies $U_K A = -A - \mathbb{1} - U_K = AU_K$. Then we write (5.10) in the form

$$G_K|s\rangle = (1-b)A|s\rangle + \frac{b}{2}A\, O_t|s\rangle - \frac{b}{2}U_K A\, O_t|s\rangle. \tag{5.11}$$

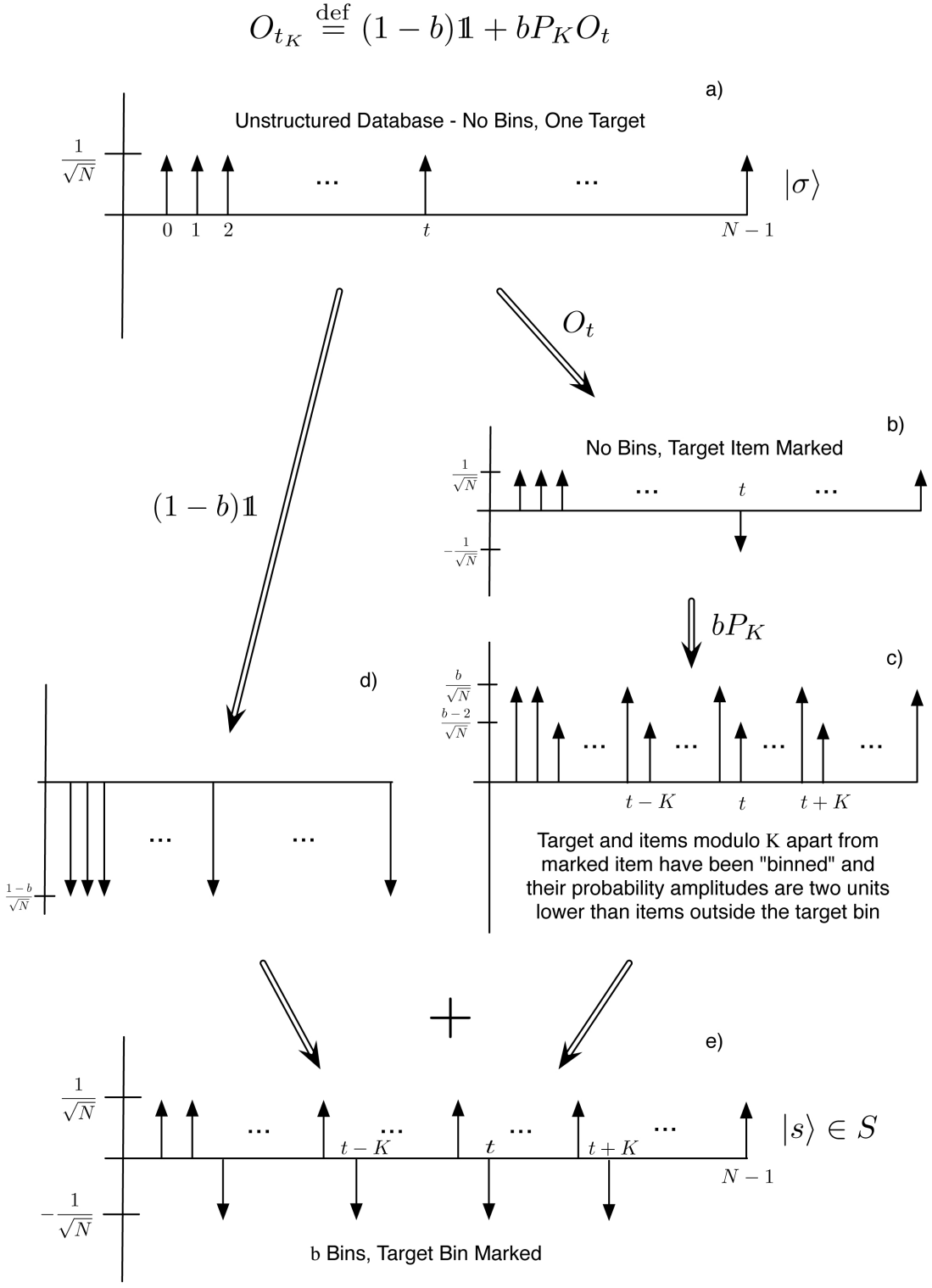

Figure 5.1: Partial Measurement Based Partial Search Oracle -

Given an input state $|\psi\rangle$, this operator generates the normalized superposition $\mathcal{N}^{-1}|\phi\rangle$ where

$$|\phi\rangle = (1-b)A|\psi\rangle + \frac{b}{2}A\,O_t|\psi\rangle - \frac{b}{2}U_K A\,O_t|\psi\rangle$$

and $\mathcal{N} = \sqrt{\langle\phi\,|\,\phi\rangle}$. This operator involves a constant number of oracle calls at each iteration, independent of the bin size. Of course, as stated earlier, G_K is only unitary in the symmetric subspace so it only works when the starting state is symmetric and it is unstable with respect to quantum noise, essentially because the space of nonsymmetric states $S^\perp$ is much larger than S, for large bin sizes b. A general state of the system has the form $|\psi\rangle = \alpha|s\rangle + \varepsilon|s^\perp\rangle$ where $\alpha^2 + \varepsilon^2 = 1$, and if this is an input at any given iteration for very small ε, then the output state of that iteration has the form

$$\mathcal{N}^{-1}(\alpha'|s'\rangle + \varepsilon(1-b)|s^\perp\rangle),$$

where $\mathcal{N}^2 = (\alpha')^2 + (1-b)^2\varepsilon^2$, and $|s'\rangle$ is a symmetric state. Hence the operator amplifies any quantum noise, specifically it amplifies the ratio of nonsymmetric to symmetric components by a factor of $b-1$, assuming that α, α' are of the same order of magnitude.

Although this instability can be controlled by means of suitable error-correcting codes (see [1] Chapter 10, for example), we propose that periodic measurement with the Hermitean operator P_K can also be used to maintain stability. The eigenvalues $0, 1$ of P_K can be regarded as comprising a "symmetry" qubit. If the state of the system has a very small nonsymmetric component then a nondemolition measurement of P_K yields 1 with near unit probability, and projects out this nonsymmetric component; the computation now proceeds correctly. If measurement of P_K happens to yield 0 then the system has been projected into a nonsymmetric state and the search must be restarted. Properties of the symmetry qubit are similar to those of the halt qubit used to determine whether a universal quantum computer has halted execution, see the discussion in [23].

In summary, the partial search algorithm performs $\frac{\pi}{4}\sqrt{K}$ iterations of G_K, beginning with $|\sigma\rangle$ as the starting state, and concludes with a full measurement which finds one of the modulo K solutions and hence determines the target bin with $n_K = O(\sqrt{K})$ standard oracle calls.

5.4.3 Implications and Problems With G_K

A sequence of partial searches using G_K can lead to much faster searches than currently known algorithms. The strategy would be to perform a sequence of partial searches, each comprising a completed quantum computation, which trap the target item in successively smaller bins. Eventually the target bin would be sufficiently small that we either find the target item quickly with a single Grover search or else, in order to minimize the number of oracle calls, reduce the target bin further to a single item. There would be a trade-off between the number of measurements, i.e. the number of partial searches, and the total number of oracle calls. In each partial search there is a less than unit probability of measuring the correct target bin, however the Grover algorithm and hence the partial search can be modified to ensure that the target bin is identified with full certainty at each stage, as discussed in [50].

Let us consider for example a 2-stage search in which the database of size N is divided into $K = \sqrt{N}$ equal blocks each of bin size $b = \sqrt{N}$. We first perform a partial search on the full database, which identifies the bin of size $\sqrt{N}$ in which the target item is located using $O(\sqrt{K}) = O(N^{1/4})$ oracle calls. Then we perform a full Grover search on this bin, which locates the target itself and also takes $O(N^{1/4})$ oracle calls. The full search is completed therefore with two measurements and $O(N^{1/4})$ oracle calls. An m-stage search following this strategy requires m measurements and $O(mN^{1/(2m)})$ oracle calls, with the number of blocks at each stage given by $K = N^{1/m}$. This search strategy applies when the number of measurements is set to a fixed number m, independent of N.

We can conduct a much faster full search by applying sufficient partial searches to reduce the target bin to a single item. After m successive partial searches of the database of size $N = 2^n$, each with fixed $K = 2^k$ independent of N, the target bin is reduced in size to $b = 2^{n-mk}$. This requires mn_K oracle calls and m measurements. Each measurement, being a classical operation, is much slower than an oracle call. Let us suppose that a measurement is equivalent in speed to M oracle calls where M is independent of N, then we have a total of $m(n_K + M)$ equivalent oracle calls, and since $n_K = O(\sqrt{K})$ we can optimize the search by choosing $K = O(M^2)$ so that the total number of equivalent oracle calls becomes $O(mM)$.

Taking now $m = n/k$, where k is independent of n, we require therefore $O(\log N)$ equivalent oracle calls in order to isolate the target, representing an exponential speedup over a full Grover search. This speedup arises because at each stage the partial search algorithm eliminates $N - b$ items from the database, a number which is proportional to N, whereas K and hence the number $O(\sqrt{K})$ of oracle calls is independent of N.

If this sounds too remarkable to be true, it is because there is a fundamental problem with G_K. To see this, let $|\psi\rangle$ be any normalized input state and we re-write equation (5.11) as

$$|\phi\rangle = (1-b)|\psi_1\rangle + \frac{b}{2}|\psi_2\rangle - \frac{b}{2}|\psi_3\rangle.$$

In order to compute $G_K|\psi\rangle$ with only two oracle calls, we would need to carry out a sequence of operations that involve calculating the nonorthogonal, normalized states $|\psi_1\rangle = A|\psi\rangle$, $|\psi_2\rangle = A\,O_t|\psi\rangle$ and $|\psi_3\rangle = U_K A\,O_t|\psi\rangle$, all of which are functions of the normalized input state $|\psi\rangle$. Thus, we would need three copies of $|\psi\rangle$ and a way to deterministically superimpose them in the same subspace. This is not possible in linear Quantum Mechanics and thus cannot realize an exponentially fast search algorithm based on this scheme unless non-linear quantum processes are ever observed. If non-linearity were to provide a way to copy quantum states, it might be possible to use such processes to realize the scheme we developed and thereby realize exponentially fast search algorithms. In fact, non-linear quantum processes would likely give rise to many superior quantum algorithms such as discussed in [52].

5.5 Two Partial Search Algorithms

In order to avoid the above discussed problem, we consider two algorithms based on the partial search oracle developed here whereby the number of oracle calls is well defined and copying of states is not required. First of all, consider the generalized form of the O_{t_K} operator that we defined in (5.2)

$$O_{t_K} = \alpha \mathbb{1} + \beta P_K O_t. \tag{5.12}$$

In (5.9), we set $\alpha = (1-b)$ and $\beta = b$. Now, even if we avoid the copying of intermediate states, a problem arises from the first term α being non-zero as this forces the normalization constant $\mathcal{N}$ that was discussed earlier to be a function of the oracle operator O_t if the input or intermediate states contain non-symmetric components. Therefore,

we can set $\alpha = 0$ and $\beta = 1$ such that the number of oracle calls is well defined and is one per O_{t_K} call. We thus obtain the non-unitary oracle

$$O_{t_K} = P_K O_t. \tag{5.13}$$

This gives rise to a partial search algorithm where we perform $n = O(\sqrt{K})$ iterations of AO_{t_K}. Noting that $[P_K, A] = 0$, $AO_{t_K} = P_K A O_t \equiv P_K G$, so we will refer to this as the $(P_K G)^n$ algorithm.

5.5.1 $(P_K G)^n$ Algorithm

For notational simplicity, we redefine $G_K \stackrel{\text{def}}{=} P_K G$, where $G \stackrel{\text{def}}{=} AO_t$. It can be readily verified that $G_K|s_i\rangle$ and $G_K|s_t\rangle_K$ can both be expressed as linear combinations of $|s_i\rangle$ and $|s_t\rangle_K$. Therefore, we assume that

$$G_K^n|\sigma\rangle = \alpha_n \sum_{i=0}^{K-1} |s_i\rangle + \beta_n |s_t\rangle_K. \tag{5.14}$$

The initial conditions require $\alpha_0 = \frac{1}{\sqrt{K}}$ and $\beta_0 = 0$. Then

$$\begin{aligned}
G_K^{n+1}|\sigma\rangle &= \alpha_n \sum_i G_K|s_i\rangle + \beta_n G_K|s_t\rangle_K \\
&= \alpha_n \sum_i A\left(|s_i\rangle - \frac{2}{b}\delta_{i,t_K}|s_t\rangle_K\right) + \beta_n\left(1 - \frac{2}{b}\right)A|s_t\rangle_K \\
&= \alpha_n \sum_i \left(\frac{2}{K}\sum_j |s_j\rangle - |s_i\rangle\right) - \frac{2\alpha_n}{b}A|s_t\rangle_K + \beta_n\left(1 - \frac{2}{b}\right)A|s_t\rangle_K \\
&= \frac{2\alpha_n}{K}\sum_{i,j}|s_j\rangle - \alpha_n\sum_i |s_i\rangle - \left(\frac{2\alpha_n}{b} - \left(1 - \frac{2}{b}\right)\beta_n\right)\left(\frac{2}{K}\sum_j |s_j\rangle - |s_t\rangle_K\right) \\
&= \left(\alpha_n - \left(\frac{2\alpha_n}{b} - \left(1 - \frac{2}{b}\right)\beta_n\right)\frac{2}{K}\right)\sum_i |s_i\rangle + \left(\frac{2\alpha_n}{b} - \left(1 - \frac{2}{b}\right)\beta_n\right)|s_t\rangle_K.
\end{aligned}$$

Thus, proving by induction that (5.14) is true. We can now solve for α_n and β_n using the recurrence relations

$$\begin{aligned}
\alpha_{n+1} &= \left(1 - \frac{4}{N}\right)\alpha_n + \left(1 - \frac{2}{b}\right)\frac{2}{K}\beta_n, \\
\beta_{n+1} &= \left(1 - \frac{2}{b}\right)\beta_n - \frac{2\alpha_n}{b}.
\end{aligned}$$

Using these equations, we obtain the characteristic equation

$$\alpha_{n+2} - \left(2 - \frac{2}{b} - \frac{4}{N}\right)\alpha_{n+1} + \left(1 - \frac{2}{b}\right)\alpha_n = 0.$$

Solving this, we get the solution

$$\begin{aligned} \alpha_n &= d_1\gamma_1^n + d_2\gamma_2^n, \\ \beta_n &= \frac{\frac{K}{2}(\alpha_{n+1} - \alpha_n) + \frac{2\alpha_n}{b}}{\left(1 - \frac{2}{b}\right)}, \end{aligned}$$

where

$$\begin{aligned} \gamma_1 &= 1 - \frac{2}{N} - \frac{1}{b} - \sqrt{\frac{4}{N}\left(\frac{1}{N} + \frac{1}{b} - 1\right) + \frac{1}{b^2}}, \\ \gamma_2 &= 1 - \frac{2}{N} - \frac{1}{b} + \sqrt{\frac{4}{N}\left(\frac{1}{N} + \frac{1}{b} - 1\right) + \frac{1}{b^2}}, \\ d_1 &= \frac{1}{2\sqrt{K}} + \frac{2b - N}{2\sqrt{N(\frac{N^2}{b} + 4N - 4bN + 4b)}}, \\ d_2 &= \frac{1}{2\sqrt{K}} - \frac{2b - N}{2\sqrt{N(\frac{N^2}{b} + 4N - 4bN + 4b)}}, \end{aligned}$$

It can be readily confirmed that this solution satisfies the initial conditions of $\alpha_0 = \frac{1}{\sqrt{K}}$ and $\beta_0 = 0$.

Now, we define the probability of success as the probability of measuring the target bin as a function of the number of iterations n and the bin size b as

$$P_{\text{success}}(n, b) \stackrel{\text{def}}{=} \left|{}_K\langle s_t | G_K^n | \sigma\rangle\right|^2,$$

and using (5.14) we obtain

$$P_{\text{success}}(n, b) = \left|\alpha_n + \beta_n\right|^2.$$

While the number of oracle calls is now well defined, it can be readily verified that

$$P_{\text{success}} = \frac{b}{N}\left(1 - \frac{2}{b}\right)^{2n} + O\left(\frac{1}{N^2}\right) \tag{5.15}$$

Thus, P_{success} is extremely low for any practical choices of N and b because it decreases exponentially with n (recall that $N = 2^n$) .

Conceptually, the pitfall with this algorithm is that at each iteration we need to restart from scratch if P_K measures a non-symmetric state. Moreover, even if P_K measures a symmetric state, it may be a non-target bin state $|s_{\neq t}\rangle_K$. In either case, the probability of measuring the target bin at the end is significantly reduced.

5.5.2 $P_K G^n$ Algorithm

This suggests another algorithm whereby we defer the application of the P_K operator until the very end. That is, we will perform n iterations of the standard full Grover search G to avoid the risk of measuring a non-symmetric or non-target bin at each iteration and instead apply P_K as the last search step. Deferring measurement operations in quantum computations is known to be possible (see [1], for example). However, we will see that while it is possible to defer measurements, the operations (i.e. the quantum circuit) prior to the deferred measurement cannot, in general, be identical in both cases. That is, measurements can indeed be deferred but the intermediate operations generally require modifications in order to do so. Conceptually, one can think of this as deferring a conditional operation in the middle of an algorithm. This is certainly possible to do but requires maintaining two branches of calculations until the deferred conditional operation is performed at the end.

In this algorithm, we apply a full Grover search for n iterations and then apply the symmetrization operator P_K as the last step. The hope is that this will give a reasonable probability of measuring the target bin $|s_t\rangle_K$.

It can be readily verified that

$$G|\sigma\rangle = \left(1 - \frac{4}{N}\right)|\sigma\rangle + \frac{2}{\sqrt{N}}|t\rangle$$

and that

$$G|t\rangle = -\frac{2}{\sqrt{N}}|\sigma\rangle + |t\rangle.$$

Thus, both $G|\sigma\rangle$ and $G|t\rangle$ can be expressed as linear combinations of $|\sigma\rangle$ and $|t\rangle$. Therefore, we assume that

$$G^n|\sigma\rangle = a_n|\sigma\rangle + b_n|t\rangle. \tag{5.16}$$

The initial conditions require $a_1 = \left(1 - \frac{4}{N}\right)$ and $b_1 = \frac{2}{\sqrt{N}}$, and note that $a_1 + b_1^2 = 1$. Now, we let

$$\begin{aligned} G^{n+1}|\sigma\rangle &= a_n G|\sigma\rangle + b_n G|t\rangle \\ &= (a_n a_1 - b_n b_1)|\sigma\rangle + (a_n b_1 + b_n)|t\rangle. \end{aligned}$$

Thus, proving by induction that (5.16) is true. We can now solve for a_n and b_n using the recurrence relations

$$\begin{aligned} a_{n+1} &= a_n a_1 - b_n b_1, \\ b_{n+1} &= a_n b_1 + b_n. \end{aligned}$$

Using these equations, we obtain the characteristic equation

$$a_{n+2} - (a_1 + 1)a_{n+1} + a_n = 0.$$

Solving this, we get the following solution to the recurrence relations

$$\begin{aligned} a_n &= c_1 \alpha_1^n + c_2 \alpha_2^n, \\ b_n &= \frac{a_n a_1 - a_{n+1}}{b1}, \end{aligned}$$

where

$$\begin{aligned} \alpha_1 &= 1 - \frac{2}{N}\left(1 - i\sqrt{N-1}\right), \\ \alpha_2 &= 1 - \frac{2}{N}\left(1 + i\sqrt{N-1}\right), \\ c_1 &= \frac{1 - N - i\sqrt{N-1}}{2 - 2N}, \\ c_2 &= \frac{1 - N + i\sqrt{N-1}}{2 - 2N}. \end{aligned}$$

It can be readily confirmed that this solution satisfies the initial conditions of $a_1 = \left(1 - \frac{4}{N}\right)$ and $b_1 = \frac{2}{\sqrt{N}}$.

As before, we define the probability of success as the probability of measuring the target bin as a function of the number of iterations n and the bin size b as follows.

$$P_{\text{success}}(n, b) \stackrel{\text{def}}{=} |_K\langle s_t | P_K G^n | \sigma \rangle|^2 .$$

Noting that $P_K|\sigma\rangle = |\sigma\rangle$, it can be readily verified that

$$P_K G^n |\sigma\rangle = a_n |\sigma\rangle + b_n P_K |t\rangle,$$

and applying P_K to $|t\rangle$ yields

$$P_K G^n |\sigma\rangle = a_n |\sigma\rangle + \frac{b_n}{\sqrt{b}} |s_t\rangle_K.$$

Therefore,

$$P_{\text{success}}(n, b) = \left| \frac{a_n}{\sqrt{K}} + \frac{b_n}{\sqrt{b}} \right|^2.$$

It can be shown that

$$P_{\text{success}} = \frac{(b + 2n)^2}{Nb} + O\left(\frac{1}{N^2}\right). \tag{5.17}$$

Hence, this algorithm also has a very low probability of success for any practical choices of N and b because it again decreases exponentially with n.

Conceptually, this is because the deferred P_K essentially "spreads" out the probability amplitude of the target state $|t\rangle$ across the members of the target bin $|s_t\rangle_K$ and since we are doing less than the optimal number of full Grover search iterations ($O(\sqrt{K})$ vs. $O(\sqrt{N})$) the probability of measuring the target bin becomes very small for $K \gg 1$. Even if we carried out a full Grover search with $n = O(\sqrt{N})$ iterations, the probability of measuring the target bin with P_K would be $O(\frac{1}{K})$ since we would be "spreading" the probability amplitude of $|t\rangle$ to the K members of $|s_t\rangle_K$.

By comparing $P_K G^2 |\sigma\rangle$ and $(P_K G)^2 |\sigma\rangle$ we immediately see that the two algorithms are not equivalent. This means that P_K cannot be deferred without also modifying the operations prior to the deferred measurement. Thus, $P_K G^n$ and $(P_K G)^n$ are not equivalent algorithms.

5.6 Discussion

We have explored a partial search algorithm based on partial measurements of symmetric states. We have created a partial search oracle based on the standard Grover oracle and involving symmetry projections. When restricted to the subspace of symmetric states, this oracle has a much simpler form that involves a simple product of the standard Grover oracle. Furthermore, this oracle acts entirely within the subspace of symmetric states. This oracle gives rise to a symmetry based partial search scheme, G_K, but difficulties in implementing a quantum circuit prevents its realization. This is not surprising because if such a quantum circuit could be implemented, it would give

rise to an exponentially fast search algorithm that would violate well known bounds for Grover search algorithms. We have also explored two closely related algorithms that lend themselves to quantum circuit implementations but these deliver no improvement over existing search algorithms.

It should be noted that the difficulty in implementing G_K and the fact that the two algorithms considered here do not yield fast results does not necessarily preclude the existence of a fast partial search algorithm based on the idea of the symmetrization or partial measurement operator. However, it would likely require a more sophisticated "guided" operator based on partial measurements at each iteration of the algorithm. In other words, a partial measurement scheme similar to the $\mathcal{UQC}$'s non-demolition measurement of the halt qubit may allow for a more efficient partial search algorithm. Furthermore, if non-linear quantum processes are ever observed, and if non-linearity were to provide a way to copy quantum states and deterministically superimpose them in the required manner, it would be possible to use such processes to implement the scheme we developed to realize exponentially fast search algorithms.

6

Conclusions And Discussion

In this thesis, we have explored two core areas of quantum computing: universal quantum computation and quantum search algorithms. The primary results are a) the construction of a universal quantum computer, b) implementation of well-known oracle based algorithms using this universal quantum computer with externally connected quantum oracle devices, c) the construction of a partial search oracle based on symmetric states and the standard Grover search oracle, and d) implementation of partial search algorithms based on this partial search oracle.

Quantum computation ultimately rests on the foundation that Deutsch laid when he devised the UQTM. After all, the power of computation, whether it be classical or otherwise, arises from programmability. Thus, the existence of a class of machines that can compute any computable algorithm as efficiently as any other machine is of utmost importance. It turned out, however, that there were fundamental and unanswered questions regarding the validity of the UQTM. While the quantum computing community at large may have deemed the raised objections as technicalities that would be eventually resolved, it nevertheless left an unpalatable gap in the foundation of quantum computation. This is the core problem that we have resolved in this thesis. We have shown by construction the existence of a universal quantum computer, $\mathcal{UQC}$, in the spirit of the UQTM.

The quantum computer that we have defined is universal in the sense that the same machine, under the control of quantum programs, can firstly emulate any classical Turing machine by being able to compute the NAND function and secondly can approximate any unitary operation to any desired accuracy by being able to apply the

set of $\{\mathrm{H}, \mathrm{CNOT}, \mathrm{T}\}$ operations on a specified set of qubits. The machine also supports conditional branching and hence conditional execution, a feature that is not directly possible in the quantum gate array circuit framework. The halting scheme that we have defined works in a way that prevents changes to the memory tape once the program has halted thus satisfying Ozawa's proof requirement and allowing for a valid program concatenation scheme. Because of its universality, $\mathcal{UQC}$ serves as a prototypical model for general-purpose programmable quantum computation and should find uses in the development and analysis of quantum algorithms and complexity. The $\mathcal{UQC}$ should also find applications in the analysis of fundamental quantum computing questions such as the exploration of the physical basis of the halting problem that we have briefly explored in this thesis.

The $\mathcal{UQC}$ is a theoretical construction. As such, we have not specified how one would go about physically implementing an actual instance of it. However, in this thesis we have also shown that the harmonic oscillator can be used to implement a universal set of quantum gates and thus could be used as the basis for a physical construction of the $\mathcal{UQC}$. The harmonic oscillator has intrinsic properties that could have advantages over spin-$\frac{1}{2}$ systems in real implementations that would need to take practical issues into account. Moreover, the harmonic oscillator energy eigenstates span an infinite dimensional Hilbert space unlike spin-$\frac{1}{2}$ systems. Although we did not explore this aspect in this thesis, it is conceivable that the infinite dimensional Hilbert space could result in some useful properties that are absent in spin-$\frac{1}{2}$ systems.

Quantum networks which connect quantum systems and can transmit quantum information are actively being investigated. Quantum connectivity provides a means of overcoming size-scaling and error-correction problems, and has significant advantages over classical connectivity. Furthermore, networks of quantum computers have also been proposed where information can be exchanged between nodes via quantum and classical channels. A general question arises as to how such quantum computers can communicate and exchange information. In the simplest case a quantum computer may download data sets from other nodes over the quantum network, but in more complex cases use the network to call subroutines, or concatenate programs from other quantum computers.

We have explored this area by means of a practical application of the $\mathcal{UQC}$ whereby we have implemented a scheme to enable it to utilize networked quantum resources. We

have illustrated the scheme by constructing $\mathcal{UQC}$ programs that implement the well-known oracle based Deutsch, Deutsch-Jozsa, and Grover algorithms using networked quantum oracle devices. We therefore have demonstrated how universal quantum computers can access networked quantum devices in a way analogous to that by which classical computers access network resources. At the same time, we have illustrated how to implement practical and useful algorithms on the $\mathcal{UQC}$.

Algorithms to search an unstructured database are extremely important because many difficult problems can be mapped to the problem of finding a specific item from a large number of possible items. In fact, if a quantum algorithm to search an unstructured database that is exponentially faster than the fastest known classical search algorithm were to exist, it would mean that quantum computability exceeds classical computability. The existence of such an algorithm would allow a quantum computer to compute the class of NP problems in polynomial time.

We have explored a partial search algorithm based on partial measurements of symmetric states. We have created a partial search oracle based on the standard Grover oracle and involving symmetry projections. When restricted to the subspace of symmetric states, this oracle has a much simpler form that involves a simple product of the standard Grover oracle. Furthermore, this oracle acts entirely within the subspace of symmetric states. This oracle gives rise to a symmetry based partial search scheme, G_K, but difficulties in implementing a quantum circuit prevents its realization. This is not surprising because if such a quantum circuit could be implemented, it would give rise to an exponentially fast search algorithm that would violate well known bounds for Grover search algorithms. We have also explored two closely related algorithms that lend themselves to quantum circuit implementations but these deliver no improvement over existing search algorithms.

It should be noted that the difficulty in implementing G_K and the fact that the two algorithms considered here do not yield fast results does not necessarily preclude the existence of a fast partial search algorithm based on the idea of the symmetrization or partial measurement operator. However, it would likely require a more sophisticated "guided" operator based on partial measurements at each iteration of the algorithm. In other words, a partial measurement scheme similar to the $\mathcal{UQC}$'s non-demolition measurement of the halt qubit may allow for a more efficient partial search algorithm. Furthermore, if non-linear quantum processes are ever observed, and if non-linearity

were to provide a way to copy quantum states and deterministically superimpose them in the required manner, it would be possible to use such processes to implement the scheme we developed to realize exponentially fast search algorithms. Research into this area is beyond the scope of this thesis and is a possible area for future consideration.

Appendix A

UQC Sample Program Execution Trace

The following table traces the state of the universal quantum computer as it executes a program that swaps the first two data qubits on the memory tape (qubits $D(1)$ and $D(2)$) of the $\mathcal{UQC}$ presented in Chapter 5. We denote the program that achieves this as $|S_{1,2}\rangle$. $|S_{1,2}\rangle$ is the sequence of instructions:

$$|\mathrm{D}+1\rangle|\mathrm{D}\rightarrow 0\rangle|\mathrm{D}+1\rangle^4|\mathrm{SWAP}\rangle|\mathrm{D}+1\rangle^5|\mathrm{SWAP}\rangle|\mathrm{D}\rightarrow 0\rangle|\mathrm{D}+1\rangle^4|\mathrm{SWAP}\rangle|\mathrm{h}\rightarrow 1\rangle|\mathrm{NOP}\rangle.$$

The evolution of $\mathcal{U}$ for program $|S_{1,2}\rangle$ is governed by equation (3.13) and proceeds as shown in Table A.1. Note that the "Operator" column denotes the transformation that is applied to the previous row. Stated differently, the columns for a given row are the results of applying the operator in that step to the state of the previous step. For example, Step 1 is the result of applying U_{IF} to the state of $\mathcal{U}$ in Step 0. Also note that even though the encoding for the NOP instruction is 0, the NOP instruction label is only used where it is interpreted as a program instruction. Thus, the state of the I register in the initialized, RI, UP, and CF states is shown as "0" instead of as NOP. The same applies to unused memory tape qubits. In the interest of keeping the table uncluttered, we omit the program instructions because they are unafftected by the evolution of $\mathcal{U}$ and only show the non-zero history and data qubits used by the program. These will be denoted $|M\rangle_H$, $|D\rangle_1$, and $|D\rangle_2$, respectively. Blank table cells indicate that there are no changes from the previous value. Finally, we use $|N\rangle$ and $|L\rangle$ to denote the F register NEXT and LOOP states, respectively.

Table A.1: $S_{1,2}$ Execution Trace

Step	Operator	$\|h\rangle$	$\|x\rangle$	$\|D\rangle$	$\|P\rangle$	$\|F\rangle$	$\|H\rangle$	$\|s\rangle$	$\|I\rangle$	$\|M\rangle_H\|D\rangle_1\|D\rangle_2$	Description
0	N/A	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|00\rangle$	$\|-1\rangle$	$\|0\rangle$	$\|0\rangle$	$\|D(1)\rangle\|D(2)\rangle$	Initial State
1	U_{IF}				$\|4\rangle$				$\|\mathrm{D}+1\rangle$		Fetch first instruction
2	U_{XD}										Point x to $M(D)$
3	U_{EX}		$\|1\rangle$								Increment x ($x \to 1$)
4	U_{UD}		$\|0\rangle$	$\|1\rangle$							Update D with x
5	U_{UF}					$\|\mathrm{N}\rangle$					Update F
6	U_{RI}				$\|0\rangle$				$\|0\rangle$		Restore I to memory tape
7	U_{UP}				$\|5\rangle$						$P \to$ next instruction
8	U_{CF}					$\|00\rangle$	$\|-3\rangle$			$\|\mathrm{N}\rangle\|D(1)\rangle\|D(2)\rangle$	Clear F
9	U_{IF}		$\|0\rangle^{[*]}$		$\|9\rangle$				$\|\mathrm{D}\to 0\rangle$		Fetch next instruction
10	U_{XD}		$\|1\rangle$	$\|0\rangle$							Point x to $M(D)$
11	U_{EX}		$\|0\rangle$								Decrement x ($x \to 0$)
12	U_{UD}										Update D with x
13	U_{UF}					$\|\mathrm{N}\rangle$					Update F
14	U_{RI}				$\|5\rangle$				$\|0\rangle$		Restore I to memory tape
15	U_{UP}				$\|10\rangle$						$P \to$ next instruction
16	U_{CF}					$\|00\rangle$	$\|-5\rangle$			$\|\mathrm{N}\rangle^2\|D(1)\rangle\|D(2)\rangle$	Clear F
17	U_{IF}				$\|14\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
18	U_{XD}										Point x to $M(D)$
19	U_{EX}		$\|1\rangle$								Increment x ($x \to 1$)
20	U_{UD}		$\|0\rangle$	$\|1\rangle$							Update D with x
21	U_{UF}					$\|\mathrm{N}\rangle$					Update F
22	U_{RI}				$\|10\rangle$				$\|0\rangle$		Restore I to memory tape
23	U_{UP}				$\|15\rangle$						$P \to$ next instruction

Step	Operator	$\|h\rangle$	$\|x\rangle$	$\|D\rangle$	$\|P\rangle$	$\|F\rangle$	$\|H\rangle$	$\|s\rangle$	$\|I\rangle$	$\|M\rangle_H\|D\rangle_1\|D\rangle_2$	Description
24	U_{CF}					$\|00\rangle$	$\|-7\rangle$			$\|\mathrm{N}\rangle^3\|D(1)\rangle\|D(2)\rangle$	Clear F
25	U_{IF}				$\|19\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
26	U_{XD}		$\|1\rangle$	$\|0\rangle$							Point x to $M(D)$
27	U_{EX}		$\|2\rangle$								Increment x ($x \to 2$)
28	U_{UD}		$\|0\rangle$	$\|2\rangle$							Update D with x
29	U_{UF}					$\|\mathrm{N}\rangle$					Update F
30	U_{RI}				$\|15\rangle$				$\|0\rangle$		Restore I to memory tape
31	U_{UP}				$\|20\rangle$						$P \to$ next instruction
32	U_{CF}					$\|00\rangle$	$\|-9\rangle$			$\|\mathrm{N}\rangle^4\|D(1)\rangle\|D(2)\rangle$	Clear F
33	U_{IF}				$\|24\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
34	U_{XD}		$\|2\rangle$	$\|0\rangle$							Point x to $M(D)$
35	U_{EX}		$\|3\rangle$								Increment x ($x \to 3$)
36	U_{UD}		$\|0\rangle$	$\|3\rangle$							Update D with x
37	U_{UF}					$\|\mathrm{N}\rangle$					Update F
38	U_{RI}				$\|20\rangle$				$\|0\rangle$		Restore I to memory tape
39	U_{UP}				$\|25\rangle$						$P \to$ next instruction
40	U_{CF}					$\|00\rangle$	$\|-11\rangle$			$\|\mathrm{N}\rangle^5\|D(1)\rangle\|D(2)\rangle$	Clear F
41	U_{IF}				$\|29\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
42	U_{XD}		$\|3\rangle$	$\|0\rangle$							Point x to $M(D)$
43	U_{EX}		$\|4\rangle$								Increment x ($x \to 4$)
44	U_{UD}		$\|0\rangle$	$\|4\rangle$							Update D with x
45	U_{UF}					$\|\mathrm{N}\rangle$					Update F
46	U_{RI}				$\|25\rangle$				$\|0\rangle$		Restore I to memory tape
47	U_{UP}				$\|30\rangle$						$P \to$ next instruction
48	U_{CF}					$\|00\rangle$	$\|-13\rangle$			$\|\mathrm{N}\rangle^6\|D(1)\rangle\|D(2)\rangle$	Clear F
49	U_{IF}			$\|4\rangle$	$\|34\rangle$				$\|\mathrm{SWAP}\rangle$		Fetch next instruction

Step	Operator	$\|h\rangle$	$\|x\rangle$	$\|D\rangle$	$\|P\rangle$	$\|F\rangle$	$\|H\rangle$	$\|s\rangle$	$\|I\rangle$	$\|M\rangle_H\|D\rangle_1\|D\rangle_2$	Description
50	U_{XD}		$\|4\rangle$	$\|0\rangle$							Point x to $M(D)$
51	U_{EX}		$\|4\rangle$	$\|0\rangle$				$\|D(1)\rangle$		$\|\mathrm{N}\rangle^6\|0\rangle\|D(2)\rangle$	Swap $D(1)$ and s
52	U_{UD}		$\|0\rangle$	$\|4\rangle$							Update D with x
53	U_{UF}					$\|\mathrm{N}\rangle$					Update F
54	U_{RI}				$\|30\rangle$				$\|0\rangle$		Restore I to memory tape
55	U_{UP}				$\|35\rangle$						$P \rightarrow$ next instruction
56	U_{CF}					$\|00\rangle$	$\|-15\rangle$			$\|\mathrm{N}\rangle^7\|0\rangle\|D(2)\rangle$	Clear F
57	U_{IF}				$\|39\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
58	U_{XD}		$\|4\rangle$	$\|0\rangle$							Point x to $M(D)$
59	U_{EX}		$\|5\rangle$								Increment x ($x \rightarrow 5$)
60	U_{UD}		$\|0\rangle$	$\|5\rangle$							Update D with x
61	U_{UF}					$\|\mathrm{N}\rangle$					Update F
62	U_{RI}				$\|35\rangle$				$\|0\rangle$		Restore I to memory tape
63	U_{UP}				$\|40\rangle$						$P \rightarrow$ next instruction
64	U_{CF}					$\|00\rangle$	$\|-17\rangle$			$\|\mathrm{N}\rangle^8\|0\rangle\|D(2)\rangle$	Clear F
65	U_{IF}				$\|44\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
66	U_{XD}		$\|5\rangle$	$\|0\rangle$							Point x to $M(D)$
67	U_{EX}		$\|6\rangle$								Increment x ($x \rightarrow 6$)
68	U_{UD}		$\|0\rangle$	$\|6\rangle$							Update D with x
69	U_{UF}					$\|\mathrm{N}\rangle$					Update F
70	U_{RI}				$\|40\rangle$				$\|0\rangle$		Restore I to memory tape
71	U_{UP}				$\|45\rangle$						$P \rightarrow$ next instruction
72	U_{CF}					$\|00\rangle$	$\|-19\rangle$			$\|\mathrm{N}\rangle^9\|0\rangle\|D(2)\rangle$	Clear F
73	U_{IF}				$\|49\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
74	U_{XD}		$\|6\rangle$	$\|0\rangle$							Point x to $M(D)$
75	U_{EX}		$\|7\rangle$								Increment x ($x \rightarrow 7$)

Step	Operator	$\|h\rangle$	$\|x\rangle$	$\|D\rangle$	$\|P\rangle$	$\|F\rangle$	$\|H\rangle$	$\|s\rangle$	$\|I\rangle$	$\|M\rangle_H\|D\rangle_1\|D\rangle_2$	Description
76	U_{UD}		$\|0\rangle$	$\|7\rangle$							Update D with x
77	U_{UF}					$\|\mathrm{N}\rangle$					Update F
78	U_{RI}				$\|45\rangle$				$\|0\rangle$		Restore I to memory tape
79	U_{UP}				$\|50\rangle$						$P \rightarrow$ next instruction
80	U_{CF}					$\|00\rangle$	$\|-21\rangle$			$\|\mathrm{N}\rangle^{10}\|0\rangle\|D(2)\rangle$	Clear F
81	U_{IF}				$\|54\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
82	U_{XD}		$\|7\rangle$	$\|0\rangle$							Point x to $M(D)$
83	U_{EX}		$\|8\rangle$								Increment x ($x \rightarrow 8$)
84	U_{UD}		$\|0\rangle$	$\|8\rangle$							Update D with x
85	U_{UF}					$\|\mathrm{N}\rangle$					Update F
86	U_{RI}				$\|50\rangle$				$\|0\rangle$		Restore I to memory tape
87	U_{UP}				$\|55\rangle$						$P \rightarrow$ next instruction
88	U_{CF}					$\|00\rangle$	$\|-23\rangle$			$\|\mathrm{N}\rangle^{11}\|0\rangle\|D(2)\rangle$	Clear F
89	U_{IF}				$\|59\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
90	U_{XD}		$\|8\rangle$	$\|0\rangle$							Point x to $M(D)$
91	U_{EX}		$\|9\rangle$								Increment x ($x \rightarrow 9$)
92	U_{UD}		$\|0\rangle$	$\|9\rangle$							Update D with x
93	U_{UF}					$\|\mathrm{N}\rangle$					Update F
94	U_{RI}				$\|55\rangle$				$\|0\rangle$		Restore I to memory tape
95	U_{UP}				$\|60\rangle$						$P \rightarrow$ next instruction
96	U_{CF}					$\|00\rangle$	$\|-25\rangle$			$\|\mathrm{N}\rangle^{12}\|0\rangle\|D(2)\rangle$	Clear F
97	U_{IF}				$\|64\rangle$				$\|\mathrm{SWAP}\rangle$		Fetch next instruction
98	U_{XD}		$\|9\rangle$	$\|0\rangle$							Point x to $M(D)$
99	U_{EX}		$\|9\rangle$					$\|D(2)\rangle$		$\|\mathrm{N}\rangle^{12}\|0\rangle\|D(1)\rangle$	Swap $D(2)$ with s
100	U_{UD}		$\|0\rangle$	$\|9\rangle$							Update D with x
101	U_{UF}					$\|\mathrm{N}\rangle$					Update F

Step	Operator	$\|h\rangle$	$\|x\rangle$	$\|D\rangle$	$\|P\rangle$	$\|F\rangle$	$\|H\rangle$	$\|s\rangle$	$\|I\rangle$	$\|M\rangle_H\|D\rangle_1\|D\rangle_2$	Description
102	U_{RI}				$\|60\rangle$				$\|0\rangle$		Restore I to memory tape
103	U_{UP}				$\|65\rangle$						$P \rightarrow$ next instruction
104	U_{CF}					$\|00\rangle$	$\|-27\rangle$			$\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
105	U_{IF}				$\|69\rangle$				$\|\mathrm{D} \rightarrow 0\rangle$		Fetch next instruction
106	U_{XD}		$\|9\rangle$	$\|0\rangle$							Point x to $M(D)$
107	U_{EX}		$\|8\rangle$								Decrement x ($x \rightarrow 8$)
108	U_{UD}		$\|0\rangle$	$\|8\rangle$				$\|D(2)\rangle$			Update D with x
109	U_{UF}					$\|\mathrm{L}\rangle$					Update F
110	U_{RI}				$\|65\rangle$				$\|0\rangle$		Restore I to memory tape
111	U_{UP}										$P \rightarrow$ next instruction
112	U_{CF}					$\|00\rangle$	$\|-29\rangle$			$\|\mathrm{L}\rangle\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
113	U_{IF}				$\|69\rangle$				$\|\mathrm{D} \rightarrow 0\rangle$		Fetch next instruction
114	U_{XD}		$\|8\rangle$	$\|0\rangle$							Point x to $M(D)$
115	U_{EX}		$\|7\rangle$								Decrement x ($x \rightarrow 7$)
116	U_{UD}		$\|0\rangle$	$\|7\rangle$							Update D with x
117	U_{UF}					$\|\mathrm{L}\rangle$					Update F
118	U_{RI}				$\|65\rangle$				$\|0\rangle$		Restore I to memory tape
119	U_{UP}										$P \rightarrow$ next instruction
120	U_{CF}					$\|00\rangle$	$\|-31\rangle$			$\|\mathrm{L}\rangle^2\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
121	U_{IF}				$\|69\rangle$				$\|\mathrm{D} \rightarrow 0\rangle$		Fetch next instruction
122	U_{XD}		$\|7\rangle$	$\|0\rangle$							Point x to $M(D)$
123	U_{EX}		$\|6\rangle$								Decrement x ($x \rightarrow 6$)
124	U_{UD}		$\|0\rangle$	$\|6\rangle$							Update D with x
125	U_{UF}					$\|\mathrm{L}\rangle$					Update F
126	U_{RI}				$\|65\rangle$				$\|0\rangle$		Restore I to memory tape
127	U_{UP}										$P \rightarrow$ next instruction

Step	Operator	$\|h\rangle$	$\|x\rangle$	$\|D\rangle$	$\|P\rangle$	$\|F\rangle$	$\|H\rangle$	$\|s\rangle$	$\|I\rangle$	$\|M\rangle_H\|D\rangle_1\|D\rangle_2$	Description
128	U_{CF}					$\|00\rangle$	$\|-33\rangle$			$\|\mathrm{L}\rangle^3\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
129	U_{IF}				$\|69\rangle$				$\|\mathrm{D}\to 0\rangle$		Fetch next instruction
130	U_{XD}		$\|6\rangle$	$\|0\rangle$							Point x to $M(D)$
131	U_{EX}		$\|5\rangle$								Decrement x ($x\to 5$)
132	U_{UD}		$\|0\rangle$	$\|5\rangle$							Update D with x
133	U_{UF}					$\|\mathrm{L}\rangle$					Update F
134	U_{RI}				$\|65\rangle$				$\|0\rangle$		Restore I to memory tape
135	U_{UP}										$P\to$ next instruction
136	U_{CF}					$\|00\rangle$	$\|-35\rangle$			$\|\mathrm{L}\rangle^4\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
137	U_{IF}				$\|69\rangle$				$\|\mathrm{D}\to 0\rangle$		Fetch next instruction
138	U_{XD}		$\|5\rangle$	$\|0\rangle$							Point x to $M(D)$
139	U_{EX}		$\|4\rangle$								Decrement x ($x\to 4$)
140	U_{UD}		$\|0\rangle$	$\|4\rangle$							Update D with x
141	U_{UF}					$\|\mathrm{L}\rangle$					Update F
142	U_{RI}				$\|65\rangle$				$\|0\rangle$		Restore I to memory tape
143	U_{UP}										$P\to$ next instruction
144	U_{CF}					$\|00\rangle$	$\|-37\rangle$			$\|\mathrm{L}\rangle^5\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
145	U_{IF}				$\|69\rangle$				$\|\mathrm{D}\to 0\rangle$		Fetch next instruction
146	U_{XD}		$\|4\rangle$	$\|0\rangle$							Point x to $M(D)$
147	U_{EX}		$\|3\rangle$								Decrement x ($x\to 3$)
148	U_{UD}		$\|0\rangle$	$\|3\rangle$							Update D with x
149	U_{UF}					$\|\mathrm{L}\rangle$					Update F
150	U_{RI}				$\|65\rangle$				$\|0\rangle$		Restore I to memory tape
151	U_{UP}										$P\to$ next instruction
152	U_{CF}					$\|00\rangle$	$\|-39\rangle$			$\|\mathrm{L}\rangle^6\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
153	U_{IF}				$\|69\rangle$				$\|\mathrm{D}\to 0\rangle$		Fetch next instruction

Step	Operator	$\|h\rangle$	$\|x\rangle$	$\|D\rangle$	$\|P\rangle$	$\|F\rangle$	$\|H\rangle$	$\|s\rangle$	$\|I\rangle$	$\|M\rangle_H\|D\rangle_1\|D\rangle_2$	Description
154	U_{XD}		$\|3\rangle$	$\|0\rangle$							Point x to $M(D)$
155	U_{EX}		$\|2\rangle$								Decrement x ($x \to 2$)
156	U_{UD}		$\|0\rangle$	$\|2\rangle$							Update D with x
157	U_{UF}					$\|\mathrm{L}\rangle$					Update F
158	U_{RI}				$\|65\rangle$				$\|0\rangle$		Restore I to memory tape
159	U_{UP}										$P \to$ next instruction
160	U_{CF}					$\|00\rangle$	$\|-41\rangle$			$\|\mathrm{L}\rangle^7\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
161	U_{IF}				$\|69\rangle$				$\|\mathrm{D} \to 0\rangle$		Fetch next instruction
162	U_{XD}		$\|2\rangle$	$\|0\rangle$							Point x to $M(D)$
163	U_{EX}		$\|1\rangle$								Decrement x ($x \to 1$)
164	U_{UD}		$\|0\rangle$	$\|1\rangle$							Update D with x
165	U_{UF}					$\|\mathrm{L}\rangle$					Update F
166	U_{RI}				$\|65\rangle$				$\|0\rangle$		Restore I to memory tape
167	U_{UP}										$P \to$ next instruction
168	U_{CF}					$\|00\rangle$	$\|-43\rangle$			$\|\mathrm{L}\rangle^8\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
169	U_{IF}				$\|69\rangle$				$\|\mathrm{D} \to 0\rangle$		Fetch next instruction
170	U_{XD}		$\|1\rangle$	$\|0\rangle$							Point x to $M(D)$
171	U_{EX}		$\|0\rangle$								Decrement x ($x \to 0$)
172	U_{UD}		$\|0\rangle$								Update D with x
173	U_{UF}					$\|\mathrm{N}\rangle$					Update F
174	U_{RI}				$\|65\rangle$				$\|0\rangle$		Restore I to memory tape
175	U_{UP}				$\|70\rangle$						$P \to$ next instruction
176	U_{CF}					$\|00\rangle$	$\|-45\rangle$			$\|\mathrm{N}\rangle\|\mathrm{L}\rangle^8\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
177	U_{IF}				$\|74\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
178	U_{XD}		$\|0\rangle$								Point x to $M(D)$
179	U_{EX}		$\|1\rangle$								Increment x ($x \to 1$)

Step	Operator	$\|h\rangle$	$\|x\rangle$	$\|D\rangle$	$\|P\rangle$	$\|F\rangle$	$\|H\rangle$	$\|s\rangle$	$\|I\rangle$	$\|M\rangle_H\|D\rangle_1\|D\rangle_2$	Description
180	U_{UD}		$\|0\rangle$	$\|1\rangle$							Update D with x
181	U_{UF}					$\|\mathrm{N}\rangle$					Update F
182	U_{RI}				$\|70\rangle$				$\|0\rangle$		Restore I to memory tape
183	U_{UP}				$\|75\rangle$						$P \rightarrow$ next instruction
184	U_{CF}					$\|00\rangle$	$\|-47\rangle$			$\|\mathrm{N}\rangle^2\|\mathrm{L}\rangle^8\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
185	U_{IF}				$\|79\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
186	U_{XD}		$\|1\rangle$	$\|0\rangle$							Point x to $M(D)$
187	U_{EX}		$\|0\rangle$	$\|2\rangle$							Increment x ($x \rightarrow 2$)
188	U_{UD}		$\|0\rangle$								Update D with x
189	U_{UF}					$\|\mathrm{N}\rangle$					Update F
190	U_{RI}				$\|75\rangle$				$\|0\rangle$		Restore I to memory tape
191	U_{UP}				$\|80\rangle$						$P \rightarrow$ next instruction
192	U_{CF}					$\|00\rangle$	$\|-49\rangle$			$\|\mathrm{N}\rangle^3\|\mathrm{L}\rangle^8\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
193	U_{IF}				$\|84\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
194	U_{XD}		$\|2\rangle$	$\|0\rangle$							Point x to $M(D)$
195	U_{EX}		$\|3\rangle$								Increment x ($x \rightarrow 3$)
196	U_{UD}		$\|0\rangle$	$\|3\rangle$							Update D with x
197	U_{UF}					$\|\mathrm{N}\rangle$					Update F
198	U_{RI}				$\|80\rangle$				$\|0\rangle$		Restore I to memory tape
199	U_{UP}				$\|85\rangle$						$P \rightarrow$ next instruction
200	U_{CF}					$\|00\rangle$	$\|-51\rangle$			$\|\mathrm{N}\rangle^4\|\mathrm{L}\rangle^8\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
201	U_{IF}				$\|89\rangle$				$\|\mathrm{D}+1\rangle$		Fetch next instruction
202	U_{XD}		$\|3\rangle$	$\|0\rangle$							Point x to $M(D)$
203	U_{EX}		$\|4\rangle$								Increment x ($x \rightarrow 4$)
204	U_{UD}		$\|0\rangle$	$\|4\rangle$							Update D with x
205	U_{UF}					$\|\mathrm{N}\rangle$					Update F

Step	Operator	$\|h\rangle$	$\|x\rangle$	$\|D\rangle$	$\|P\rangle$	$\|F\rangle$	$\|H\rangle$	$\|s\rangle$	$\|I\rangle$	$\|M\rangle_H\|D\rangle_1\|D\rangle_2$	Description
206	U_{RI}				$\|85\rangle$				$\|0\rangle$		Restore I to memory tape
207	U_{UP}				$\|90\rangle$						$P \rightarrow$ next instruction
208	U_{CF}					$\|00\rangle$	$\|-53\rangle$			$\|\mathrm{N}\rangle^5\|\mathrm{L}\rangle^8\|\mathrm{N}\rangle^{13}\|0\rangle\|D(1)\rangle$	Clear F
209	U_{IF}				$\|94\rangle$				$\|\mathrm{SWAP}\rangle$		Fetch next instruction
210	U_{XD}		$\|4\rangle$	$\|0\rangle$							Point x to $M(D)$
211	U_{EX}							$\|0\rangle$		$\|\mathrm{N}\rangle^5\|\mathrm{L}\rangle^8\|\mathrm{N}\rangle^{13}\|D(2)\rangle\|D(1)\rangle$	Swap $D(1)$ with s
212	U_{UD}		$\|0\rangle$	$\|4\rangle$							Update D with x
213	U_{UF}					$\|\mathrm{N}\rangle$					Update F
214	U_{RI}				$\|90\rangle$				$\|0\rangle$		Restore I to memory tape
215	U_{UP}				$\|95\rangle$						$P \rightarrow$ next instruction
216	U_{CF}					$\|00\rangle$	$\|-55\rangle$			$\|\mathrm{N}\rangle^6\|\mathrm{L}\rangle^8\|\mathrm{N}\rangle^{13}\|D(2)\rangle\|D(1)\rangle$	Clear F
217	U_{IF}				$\|99\rangle$				$\|\mathrm{h} \rightarrow 1\rangle$		Fetch next instruction
218	U_{XD}		$\|4\rangle$	$\|0\rangle$							Point x to $M(D)$
219	U_{EX}	$\|1\rangle$									Set the halt qubit
220	U_{UD}		$\|0\rangle$	$\|4\rangle$							Update D with x
221	U_{UF}					$\|\mathrm{N}\rangle$					Update F
222	U_{RI}				$\|95\rangle$				$\|0\rangle$		Restore I to memory tape
223	U_{UP}				$\|100\rangle$						$P \rightarrow$ next instruction
224	U_{CF}					$\|00\rangle$	$\|-57\rangle$			$\|\mathrm{N}\rangle^7\|\mathrm{L}\rangle^8\|\mathrm{N}\rangle^{13}\|D(2)\rangle\|D(1)\rangle$	Clear F
225	U_{IF}				$\|104\rangle$				$\|\mathrm{NOP}\rangle$		Fetch next instruction
226	U_{XD}		$\|4\rangle$	$\|0\rangle$							Point x to $M(D)$
227	U_{EX}										No effect
228	U_{UD}		$\|0\rangle$	$\|4\rangle$							Update D with x
229	U_{UF}					$\|\mathrm{L}\rangle$					Update F
230	U_{RI}				$\|100\rangle$				$\|0\rangle$		Restore I to memory tape
231	U_{UP}										$P \rightarrow$ next instruction

Step	Operator	$\|h\rangle$	$\|x\rangle$	$\|D\rangle$	$\|P\rangle$	$\|F\rangle$	$\|H\rangle$	$\|s\rangle$	$\|I\rangle$	$\|M\rangle_H\|D\rangle_1\|D\rangle_2$	Description
232	U_{CF}					$\|00\rangle$	$\|-59\rangle$				Clear F
233	U_{IF}				$\|104\rangle$				$\|\mathrm{NOP}\rangle$		Fetch next instruction
234	U_{XD}		$\|4\rangle$	$\|0\rangle$							Point x to $M(D)$
235	U_{EX}										No effect
236	U_{UD}		$\|0\rangle$	$\|4\rangle$							Update D with x
237	U_{UF}					$\|\mathrm{L}\rangle$					Update F
238	U_{RI}				$\|100\rangle$				$\|0\rangle$		Restore I to memory tape
239	U_{UP}										$P \rightarrow$ next instruction
240	U_{CF}					$\|00\rangle$	$\|-61\rangle$				Clear F
241	U_{IF}				$\|104\rangle$				$\|\mathrm{NOP}\rangle$		Fetch next instruction
242	U_{XD}		$\|4\rangle$	$\|0\rangle$							Point x to $M(D)$
243	U_{EX}										No effect
244	U_{UD}		$\|0\rangle$	$\|4\rangle$							Update D with x
245	U_{UF}					$\|\mathrm{L}\rangle$					Update F
246	U_{RI}				$\|100\rangle$				$\|0\rangle$		Restore I to memory tape
247	U_{UP}										$P \rightarrow$ next instruction
248	U_{CF}					$\|00\rangle$	$\|-63\rangle$				Clear F
.	.	.	.	.	.		.	.	.	.	Loop on NOP forever
.	.	.	.	.	.		.	.	.	.	.
.	.	.	.	.	.		.	.	.	.	.
.	.	.	.	.	.		.	.	.	.	.

A. UQC SAMPLE PROGRAM EXECUTION TRACE

[*] Note that even though some registers and qubits appear stationary at operator application boundaries, in actuality they may make intermediate state transitions. For example, in this particular case, $|x\rangle$ undergoes the transitions $|0\rangle \rightarrow |5\rangle \rightarrow |6\rangle \rightarrow |7\rangle \rightarrow |8\rangle \rightarrow |9\rangle \rightarrow |0\rangle$ when U_{IF} is applied due to the $|x\rangle \leftrightarrow |P\rangle$ swaps and the INC_x transformations.

Appendix B

Published Work

CONSTRUCTION OF A UNIVERSAL QUANTUM COMPUTER

Antonio A. Lagana, M. A. Lohe, Lorenz von Smekal

Department of Physics, University of Adelaide

South Australia 5005, Australia

PHYSICAL REVIEW A 79, 052322 1-11 (2009)

B. PUBLISHED WORK

STATEMENT OF AUTHORSHIP

CONSTRUCTION OF A UNIVERSAL QUANTUM COMPUTER

PHYSICAL REVIEW A 79, 052322 1-11 (2009)

ANTONIO A. LAGANA (Candidate)

Performed research of the universal quantum turing machine and the issues raised about its validity, constructed universal quantum computer by combining knowledge in quantum mechanics with my background and expertise in electrical engineering, computer science, and semiconductor chip design.

I hereby certifv that the statement of contribution is accurate.

Signed Date. 24/3/2010

M. A. LOHE

Provided expertise in quantum mechanics, reviewed research, highlighted areas for deeper explanation and focus, provided constructive criticism and better ways to present the research work.

I hereby certify that the statement of contribution is accurate and I give permission for the inclusion of the paper in the thesis.

Signed Date. 24/3/2010

LORENZ VON SMEKAL

Provided expertise in quantum mechanics, reviewed research, highlighted areas for deeper explanation and focus, provided constructive criticism and better ways to present the research work.

I hereby certify that the statement of contribution is accurate and I give permission for the inclusion of the paper in the thesis.

Signed Date. 23.3.2010

PHYSICAL REVIEW A **79**, 052322 (2009)

Construction of a universal quantum computer

Antonio A. Lagana,* M. A. Lohe, and Lorenz von Smekal
Department of Physics, University of Adelaide, South Australia 5005, Australia
(Received 30 March 2009; published 18 May 2009)

We construct a universal quantum computer following Deutsch's original proposal of a universal quantum Turing machine (UQTM). Like Deutsch's UQTM, our machine can emulate any classical Turing machine and can execute any algorithm that can be implemented in the quantum gate array framework but under the control of a quantum program, and hence is universal. We present the architecture of the machine, which consists of a memory tape and a processor and describe the observables that comprise the registers of the processor and the instruction set, which includes a set of operations that can approximate any unitary operation to any desired accuracy and hence is quantum computationally universal. We present the unitary evolution operators that act on the machine to achieve universal computation and discuss each of them in detail and specify and discuss explicit program halting and concatenation schemes. We define and describe a set of primitive programs in order to demonstrate the universal nature of the machine. These primitive programs facilitate the implementation of more complex algorithms and we demonstrate their use by presenting a program that computes the NAND function, thereby also showing that the machine can compute any classically computable function.

DOI: 10.1103/PhysRevA.79.052322 PACS number(s): 03.67.Lx

I. INTRODUCTION

As best exemplified by Shor's factorization algorithm [1], quantum computing algorithms have the potential to achieve significant speed-ups over classical computing algorithms. In fact, it has even been suggested [2] that quantum computability can potentially surpass classical computability by solving problems such as the famous halting problem.

The gate array model of quantum computing that Deutsch [3] formulated has been shown to be universal in the sense that any unitary operation can be implemented using a set of single-qubit gates (H and the T or $\frac{\pi}{8}$ gate) and the two-qubit controlled-NOT (CNOT) gate much as the two-bit NAND gate is universal for classical logical circuits (see Nielsen and Chuang [4] chapter 4, for example). This means that a quantum computer can compute any function that can be computed by a classical computer and in certain cases, most notably unstructured database search [5] and integer factorization [1], with a speed-up over the best known classical algorithms. However, this universality is distinct from the notion of universality in the sense that every computable function can be computed by a universal Turing machine (UTM). According to the Church-Turing thesis, all finitely realized computing machines can be simulated by a *single* machine, the UTM. Modern computers are fundamentally UTM implementations. Thus, universality is important because programmability follows from it.

The quantum gate array computing model is not universal in this sense because it is not programmable. A single quantum gate array computer cannot simulate every other quantum computer. Each quantum gate array computer must be purpose built or configured to implement a particular algorithm. Even the recently proposed quantum adiabatic computing framework [6] is not strictly programmable in this sense because the time-dependent Hamiltonian must be individually tailored for each given problem. In this sense, currently envisioned quantum computers more closely resemble special purpose processors rather than general-purpose processors, to draw an analogy with classical computers.

In one of the founding papers of quantum computation, Deutsch [7] defined a quantum Turing machine (QTM) and further claimed that there exists a universal quantum Turing machine (UQTM) that is a natural quantum generalization of the classical UTM, both of which are quantum generalizations of their classical counterparts. The UQTM was defined to be a QTM for which there exists a program as part of its input state that has the effect of applying a unitary transformation on an arbitrary number of qubits arbitrarily close to any desired transformation. That is, the UQTM could simulate, up to arbitrary accuracy, the operation of any given QTM. As its classical counterpart, a QTM contains a halt qubit that is used to indicate whether the computation has been completed. Thus, the UQTM is universal in the sense in that it is programmable and able to simulate the operation of every possible QTM. The theoretical existence of such a machine is important because it would establish whether a programmable quantum computer can be constructed in principle.

Since the original proposal, several questions have been raised as to whether the UQTM, as defined by Deutsch, was indeed valid. In 1997, Myers [8] argued that the UQTM's halting scheme was invalid. In 2001 Shi [9] showed that the validity of the halting scheme ultimately rested on whether the program concatenation scheme was valid. If the concatenation scheme were valid, the halting scheme would be valid, and Myer's question could be resolved by Ozawa's [10] nondemolition measurements of the halting qubit subject to the requirement that the halting scheme be implemented in a way whereby the state of the memory tape ceases to change once the halt qubit is set. The question of whether the concatenation scheme is valid and hence whether a UQTM exists has remained an open question.

In the following sections, we first review the QTM and UQTM as defined by Deutsch and then present an explicit

*antonio.lagana@adelaide.edu.au

construction of a universal quantum computer to demonstrate that a universal (programmable) quantum computer exists and that program concatenation is valid. The machine supports programmatic execution basic instructions that include the universal set of unitary operations as well as a conditional branch instruction. Like Deutsch's UQTM, our machine consists of a memory tape and processor. The internal architecture of our machine is very similar to that of a classic microcontroller and contains a data address register, program counter, status flag, instruction fetch buffer register, and a memory read-and-write head. In addition, our machine contains a halt qubit that signifies whether program execution has completed and a flow control register and history buffer address register that are used to store program execution history information. The flow control and history buffer address registers are used to store a sufficient amount of information such that, in principle, the operation of any program can be reversed at any given time, consistent with unitarity. This theoretical construction will be useful to analyze other aspects of quantum computation, such as complexity analysis of algorithms, analysis of the halting problem (in the Church-Turing thesis sense), etc., in an analogous way that a UTM is used in classical computer science.

Sections II and III provide brief descriptions of Deutsch's QTM and UQTM, respectively. In Sec. IV we describe several problems that were raised about the QTM halting scheme and the fact that it relies on the validity of program concatenation, something that Deutsch did not prove. In Sec. V we present an explicit construction of a universal quantum computer and describe the internal architecture, instruction set, and the time evolution operator associated with the machine. In Sec. VI we define a set of basic programs in order to demonstrate the classical universal nature of our machine by constructing a program that computes the NAND function. In Sec. VII we discuss program concatenation and present a program concatenation operator for our machine thus demonstrating that program concatenation is valid for quantum computers.

II. QUANTUM TURING MACHINE

As defined by Deutsch, a QTM consists of two components: a finite processor and an infinite tape (external memory), of which only a finite portion is ever used. The finite processor consists of N qubits and the infinite tape consists of an infinite sequence of qubits, of which only a finite portion is ever used. The currently scanned tape location is specified by x which denotes the "address" number of the tape. Thus, the state of a QTM is a unit vector in the Hilbert space spanned by the basis states $|x\rangle|\mathbf{n}\rangle|\mathbf{m}\rangle$, where $|\mathbf{n}\rangle \overset{\text{def}}{=} |n_0,n_1,n_2,\ldots,n_{N-1}\rangle$, and $|\mathbf{m}\rangle \overset{\text{def}}{=} |\ldots,m_{-2},m_{-1},m_0,m_1,m_2\ldots\rangle$.

The operation or dynamics of the machine is defined by a *fixed* unitary operator U whose only nontrivial matrix elements are $\langle x\pm 1;\mathbf{n}';m_x',m_{y\neq x}|U|x;\mathbf{n};m_x,m_{y\neq x}\rangle$. That is, only one tape qubit, the xth, participates in any given computational step and at each step, the position of the head cannot change by more than one unit, forward or backward, or both in the case that the position of the tape is a superposition of $|x\pm 1\rangle$. Each different U corresponds to a *different* QTM. Stated differently, each QTM corresponds to a specific algorithm in the same way that each quantum gate array circuit is an implementation of a specific algorithm. To signal whether the computation has been completed, the processor contains a special internal qubit, $|n_0 \overset{\text{def}}{=} h\rangle$, known as the halt qubit, that is initialized to 0 and is set to 1 upon completion of the computation. Thus, an external operator (or classical computer) may periodically observe $|h\rangle$ to determine whether the computation has been completed. The evolution of the QTM can thus be described as

$$|\psi(s\Delta T)\rangle = U^s|\psi(0)\rangle,$$

where $|\psi(0)\rangle$ is the initial state, s is the number of computation steps, and ΔT is the time duration of each computational step.

III. UNIVERSAL QUANTUM TURING MACHINE

As Shi [9] pointed out, a UQTM state may be defined as $|Q,D,P,\Sigma\rangle$, where Q is the state of the processor, including the head position x, D is the state of the data register, and P is the program state. D and P are each parts of the tape and Σ is the remaining part of the tape that is not used during the computation. Note that this does not deviate from the original definition of the UQTM by Deutsch in [7], as the corresponding basis elements of $|\mathbf{m}\rangle$ can be appropriately mapped to the corresponding basis elements of D, P, and Σ.

Deutsch claimed that there is a UQTM with which is associated a special unitary transformation U that when applied a positive integer number of times can come arbitrarily close to applying any desired unitary transformation on a finite number of data qubits. Stated differently, the claim was that there exists a UQTM, i.e., a special U, so that for an arbitrary accuracy ϵ and arbitrary unitary transformation $\mathcal{U}$ which changes D to $\mathcal{U}D$, there is always a program state $P(D,\mathcal{U},\epsilon)$ and a positive integer $s=s(D,\mathcal{U},\epsilon)$, such that

$$U^s|Q,D,P,\Sigma\rangle = |Q',D',P',\Sigma\rangle,$$

where D' is arbitrarily close to $\mathcal{U}$D, i.e., $\|D'-\mathcal{U}D\|^2<\epsilon$. Finally, like the QTM, the UQTM contains a special internal halt qubit $|h\rangle$ that is monitored to determine whether the computation has completed.

IV. IS THE HALTING SCHEME VALID?

In 1997 Myers [8] suggested that the UQTM's halting scheme was invalid. He argued that an entanglement between the halt qubit and other qubits could occur, thereby making it impossible to determine whether the machine has halted or not. His reasoning was as follows: suppose that two computations, A and B, halt after N_A and N_B steps, respectively, and without loss of generality, that $N_B>N_A$. Then for a computation that is a superposition of computations A and B, after N steps of the UQTM with $N_A<N<N_B$, the halt qubit will be in a superposition of halted and not halted states due to the linearity of the quantum evolution.

Because the computation time is unknown *a priori*, measurement of the halt qubit would collapse the state of the machine to that corresponding to the intermediate computation state of B (with $|h\rangle=|0\rangle$) or to the completed computation state of A (with $|h\rangle=|1\rangle$). Myers argued that this was a conflict between being universal and "being fully quantum," i.e., that the UQTM halting scheme was incompatible with superposition and hence the machine would need to operate on classic states. Conceptually, one could argue that this is not really a problem because any program will ultimately generate a single result. The case of superposed programs corresponds to the classical case of running a program with random data. The computation result depends on the data. In the superposed quantum computer case, the result obtained depends on the final measurement probabilities for obtaining each of the superposed program results.

In 1998 Ozawa [10] showed that monitoring $|h\rangle$ is a quantum nondemolition measurement, that is, periodic measurement of $|h\rangle$ while the computation is in progress does not alter the final measurement of the memory tape contents, which store the result of the computation. This is true even if $|h\rangle$ becomes entangled with other qubits during the computation. The crucial aspect of this proof is that the probabilities of obtaining each of the possible superposed results are not altered by periodic measurement of the halt qubit. The periodic measurement could be said to collapse the machine to one of the many superposed branches of computation as Myers aptly highlighted, but the probability of measuring that particular computational branch is no different than if the measurement is postponed until after the program has completed execution. The key assumption or requirement in Ozawa's proof is that the state of the memory tape remains unchanged once the halt qubit is set.

Furthermore, in 2001 Shi [9] also highlighted that universality and "being fully quantum" does not require the *entire* UQTM to evolve from a superposition. The superposition need only be on the *data* state. For example, if the data state is $|D\rangle=|A\rangle+|B\rangle$, the state of the total system starts at $|Q,A+B,P(A+B,\mathcal{U},\epsilon),\Sigma\rangle$, rather than at $|Q,A,P(A,\mathcal{U},\epsilon),\Sigma\rangle+|Q,B,P(B,\mathcal{U},\epsilon),\Sigma\rangle$.

However, the scenario highlighted by Myer would arise if one were to require that the program be only dependent on the desired transformation $\mathcal{U}$ and the accuracy ϵ, but independent of the initial data state. In this case, a computation on data state $D=A+B$ would need to start at $|Q,A+B,P(\mathcal{U},\epsilon),\Sigma\rangle$, or $|Q,A,P(\mathcal{U},\epsilon),\Sigma\rangle+|Q,B,P(\mathcal{U},\epsilon),\Sigma\rangle$. Hence in this case entanglement between the halt qubit and the rest of the system would occur if the execution times for A and B were different, which would be generally the case. However, the requirement for a data state independent program is unnecessary and the halt qubit entanglement problem could thus be avoided. Also if we require the programs to be data state independent and the halt qubit becomes entangled, Ozawa's proof applies and periodic measurements of the halt qubit do not affect the outcome of the computation.

However, Shi also pointed out that the halting scheme is a special case of the program *concatenation* scheme that was assumed to be valid in the original UQTM proposal. The original definition of the UQTM is based on the assumption that if there is a program whose effect is to apply $\mathcal{U}$ on the data state $|D\rangle$, then there exists a unitary operator whose effect is $|h=1\rangle\langle h=0|\otimes\mathcal{U}$ on $|h=0\rangle|D\rangle$. This assumption was not proven and the validity of program concatenation has not been addressed in other work (see Sec. 8.3 in [11], for example) that relies upon the QTM defined by Bernstein and Vazirani [12] in 1997; this version of the QTM not only requires a halting scheme like Deutsch's but also requires that every computational path reach a final configuration simultaneously, and thus every computational path must be somehow synchronized.

The problem with synchronizing every computational path is that, in general, it is not known *a priori* how long a program will take to halt or if it will halt at all because program execution times can depend on the data that the program operates upon. This problem was highlighted by several authors, including Iriyama, Miyadera, and Ohya as recently as 2008 [13]. Thus, it is not always possible to find an upper bound T on the time needed for all branches to halt and thereby equip each branch of a computation with a counter that increments at each time step and halts once it reaches some upper bound T. In essence, such a synchronization scheme is well suited for dealing with sequential programs that are guaranteed to halt but not for programs that may never halt due to conditional branches or loops.

We address these open questions by constructing a theoretical universal quantum computer with valid and explicit halting and program concatenation schemes, and which also supports conditional branching and does not require synchronization of all computational paths. This machine serves as a prototypical model for a general-purpose programmable quantum computer that will be useful in the development and analysis of new quantum algorithms, complexity analysis of quantum algorithms, and investigation of the physical basis of the Turing halting problem.

V. UNIVERSAL QUANTUM COMPUTER

Our goal is to devise a quantum computer that can compute any computable function. The machine itself is to be fixed and each different function is to be computed by providing the machine with a suitable set of input data and program. Any unitary operation can be approximated to any desired accuracy using the set of {H,CNOT,T} gates (see Nielsen and Chuang [4], Chap. 4, for example), where $\mathrm{H}\overset{\mathrm{def}}{=}\frac{1}{\sqrt{2}}\{(|0\rangle+|1\rangle)\langle 0|+(|0\rangle-|1\rangle)\langle 1|\}$, $\mathrm{CNOT}\overset{\mathrm{def}}{=}|00\rangle\langle 00|+|01\rangle\langle 01|+|11\rangle\langle 10|+|10\rangle\langle 11|$, and $\mathrm{T}\overset{\mathrm{def}}{=}|0\rangle\langle 0|+e^{i\pi/4}|1\rangle\langle 1|$. This set is universal in the sense that any function (i.e., unitary operation) that can be computed by a quantum computer can be implemented using a combination of these gates. Thus, to create a universal quantum computer in the programmable sense, it suffices to devise one that can implement these operations on a specified set of qubits under the control of a quantum program. The quantum computer described below and illustrated in Fig. 1 is an instance of such a machine.

Following Deutsch [7], our machine $\mathcal{UQC}$ consists of two primary parts: a processor $\mathcal{Q}$ that implements the universal

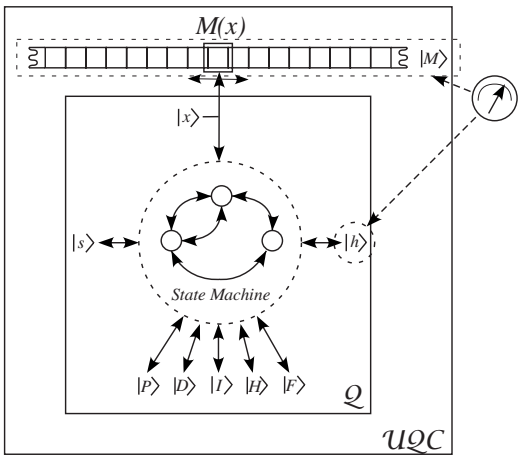

FIG. 1. Architecture of the universal quantum computer $\mathcal{UQC}$, showing the memory tape M, processor $\mathcal{Q}$, address of tape head $|x\rangle$, scratch qubit $|s\rangle$, instruction register $|I\rangle$, program address register $|P\rangle$, data address register $|D\rangle$, history address register $|H\rangle$, flow control register $|F\rangle$, halt qubit $|h\rangle$, and the qubits that are measured ($|M\rangle$ and $|h\rangle$).

set of unitary operations and an infinite tape that acts as the machine's external memory. The tape consists of an infinite sequence of qubits, $|M\rangle=\{|m_i\rangle\}, i \in \mathbb{Z}$, with only a finite subset of them ever being used for any given program. This corresponds to a classical computer's memory and external storage which, while finite, can be arbitrarily large. With the tape is associated an observable $\hat{x}$ in the processor that has the whole of $\mathbb{Z}$ as its spectrum and that acts as the address number of the currently scanned tape location. Addressing different tape locations can be realized either by a movable head over a static tape or by a movable tape under a static head. Since either scheme is identical for the purposes of constructing $\mathcal{UQC}$, we assume the latter as that allows for $\mathcal{Q}$ to be fixed in space, and movement of the tape is accomplished by a sliding "bin" of qubits that moves under $\mathcal{Q}$'s control.

As part of its internal state machine, $\mathcal{Q}$ also contains two additional observables, $\hat{D}$ and $\hat{P}$, that act as the data address and program counter, respectively. $\hat{D}$ is used to address individual data qubits on the tape and to specify the branch destination address and $\hat{P}$ is used to keep track of the program instruction that is to be executed. As with classical computers, $\hat{D}$ and $\hat{P}$ need not have an infinite spectrum as they need only be as "wide" as required to address the finite subset of the infinite tape that would ever be used. However, for the purpose of the most general construction, we do not restrict $\mathcal{UQC}$ to have a particular address range and thus treat $\hat{D}$ and $\hat{P}$ (and $\hat{x}$) as having an infinite spectrum.

$\mathcal{Q}$ also contains a four-qubit register $\hat{I}$ to load the instruction to be executed. In order to perform the two-qubit CNOT operation, $\mathcal{Q}$ contains a "scratch" qubit $|s\rangle$ that is used as the control qubit. Like Deutsch's UQTM, $\mathcal{UQC}$ also contains a dedicated observable qubit $|h\rangle$ that indicates whether the program execution has completed (i.e., the halt qubit). $\mathcal{Q}$ also contains a two-qubit register $\hat{F}$ that is used to control the execution flow (i.e., whether the program should loop on the current instruction, proceed to the next instruction, or branch to a new instruction). Finally, $\mathcal{UQC}$ contains a register $\hat{H}$ with the same spectrum as $\hat{x}$, $\hat{D}$, and $\hat{P}$. The purpose (and naming) of the $\hat{H}$ register is described later. For notational simplicity, we drop the $\hat{}$ notation hereafter when referring to $\mathcal{UQC}$ registers, e.g., D refers to the observable $\hat{D}$ whose corresponding state is $|D\rangle$.

The overall state of $\mathcal{UQC}$, then, is given by $|h,x,D,P,F,H,s,I,M\rangle$, where $|h,D,P,F,H,s,I\rangle \overset{\text{def}}{}$ corresponds to Deutsch's $|\mathbf{n}\rangle$ with $|h\rangle=|n_0\rangle$.

Each program consists of a finite sequence of four qubit instruction words. Self-modifying code is to be avoided because modifying program instructions during program execution can lead to unpredictable results. For example, the processor fetches instructions to be executed from the memory tape into the temporary internal buffer register I by swapping the contents of the memory tape and the I register (and swapping back the two when the instruction has been executed). Because I is initialized to $|0\rangle$, the swapped contents of the memory tape temporarily become $|0\rangle$ while the instruction is being executed. This means that if the program attempts to modify the location of the instruction being executed, it would be modifying $|0\rangle$ and not the actual instruction (that is temporarily held in the I register). This can lead to unintended and unpredictable behavior.

The instruction set of $\mathcal{UQC}$ is as follows. As mentioned earlier, we implement a universal set of unitary operations, namely {H,CNOT,T}, in order to ensure that $\mathcal{UQC}$ is universal. In order to enable the programmer to address any qubit on the memory tape and thus apply the universal set of operations to any qubit, we implement three instructions: an instruction to set D to 0, an instruction to increment D by 1, and an instruction to decrement D by 1. Because the CNOT operation requires two operands (control and data), we implement a swap instruction to enable the programmer to swap the qubit on the memory tape pointed to by D with the machine's s qubit, thereby enabling any qubit on the memory tape to be used as the control qubit. While not strictly necessary for universality, we implement a branching scheme in $\mathcal{UQC}$ because first, this is not explicitly possible in other popular quantum computing frameworks such as the gate array framework and second, because it is a common operation in classical computers. Branching is essentially implemented by allowing the programmer to swap the data register and program counter register contents, thereby allowing the program to branch to any instruction on the memory tape. We also implement an instruction to effectively clear s by swapping its contents with the next available 0 slot on the negative portion of the memory tape (pointed to by H). The clear s instruction provides for a simple and convenient way for the programmer to load s with 0 without having to hunt around the memory tape looking for a 0 data qubit slot. Finally, we implement an instruction to set the halt qubit to 1 but because we also want the memory tape to remain unchanged once the halt qubit is set, we implement an accompanying instruction (NOP) to follow the halt instruction that will accomplish this.

TABLE I. $\mathcal{UQC}$ instruction set.

Label	Encoding	Description
$\|\mathrm{NOP}\rangle$	$\|0000\rangle$	No operation
$\|D\rightarrow 0\rangle$	$\|0001\rangle$	$D\rightarrow 0$
$\|D+1\rangle$	$\|0010\rangle$	$D\rightarrow D+1$
$\|D-1\rangle$	$\|0011\rangle$	$D\rightarrow D-1$
$\|\mathrm{H}\rangle$	$\|0100\rangle$	Apply Hadamard operation to $\|M\rangle_D$
$\|\mathrm{T}\rangle$	$\|0101\rangle$	Apply T operation to $\|M\rangle_D$
$\|\mathrm{SWAP}\rangle$	$\|0110\rangle$	$\|M\rangle_D \leftrightarrow \|s\rangle$
$\|\mathrm{CNOT}\rangle$	$\|0111\rangle$	CNOT of $\|M\rangle_D$ and $\|s\rangle$ ($\|s\rangle$: control)
$\|D\leftrightarrow P\rangle$	$\|1000\rangle$	$\|D\rangle \leftrightarrow \|P\rangle$ (branch) if $s=0$
$\|\mathrm{CLS}\rangle$	$\|1001\rangle$	Clear s
$\|\mathrm{R}_0\rangle$	$\|1010\rangle$	Unused
$\|\mathrm{R}_1\rangle$	$\|1100\rangle$	Unused
$\|\mathrm{R}_2\rangle$	$\|1101\rangle$	Unused
$\|\mathrm{R}_3\rangle$	$\|1110\rangle$	Unused
$\|h\rightarrow 1\rangle$	$\|1111\rangle$	$\|h\rangle \rightarrow \|1\rangle$ (set halt qubit)

The instruction set of $\mathcal{UQC}$, then, consists of 11 instructions, whose operations and encodings are defined in Table I. The single qubit operations H and T act on the qubit at tape location $M(D)$, denoted $|M\rangle_D$, and the two qubit operations SWAP and NAND act on $|M\rangle_D$ and the scratch qubit $|s\rangle$, the latter being used as the control qubit for the NAND operation.

The operation of $\mathcal{UQC}$ proceeds as follows:

(1) An external operator (or classical computer) initializes the state of M at $t=0$ with the desired data and program. Data qubit i, $i \in \mathbb{Z}^+$, is placed on tape location $M(5i-1)$ and program instruction j, $j \in \mathbb{Z}^+$, is placed on tape locations $M[5j-2:5(j-1)]$, i.e., data are placed at $M(4), M(9), M(14), \ldots$, and program instructions are placed at $M(3:0), M(8:5), M(18:15), \ldots$. The negative portions of the tape are initialized to $|0\rangle$, as illustrated in Fig. 2.

(2) The processor registers are all initialized to $|0\rangle$.

(3) An external operator starts $\mathcal{Q}$ by releasing it from the reset state.

(4) $\mathcal{Q}$ fetches the program instruction at tape location $M(P)$ into register I.

(5) $\mathcal{Q}$ executes the operation specified by I.

(6) If the halt qubit $|h\rangle$ becomes set, $\mathcal{Q}$ halts execution (strictly speaking, because $\mathcal{UQC}$ is a quantum system, $\mathcal{Q}$ continues to evolve but the evolution of the memory tape becomes trivial—i.e., $U=\mathbb{1}$—after the halt qubit has been set) and awaits an external measurement of the results. Otherwise, $\mathcal{Q}$ continues execution of the program by loading the next program instruction.

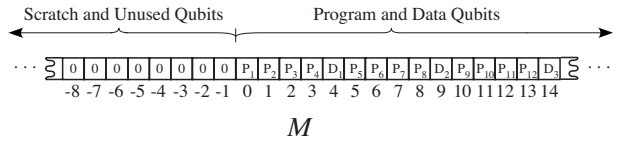

FIG. 2. Initial memory tape contents. The negative qubit slots are used as scratch qubits and the non-negative qubit slots are initialized with interleaved program instruction and data qubits.

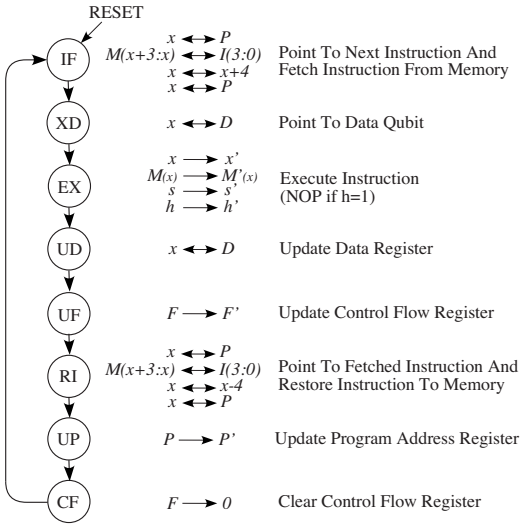

FIG. 3. $\mathcal{Q}$ state machine diagram that corresponds to the evolution of the universal quantum computer. The overall evolution is determined by eight unitary transformations.

(7) An external operator periodically performs a measurement on the halt qubit.

(8) If measurement of the halt qubit yields $|1\rangle$, the program has completed execution. The results are obtained by measuring the contents of M. Otherwise, $\mathcal{Q}$ is allowed to continue program execution.

The operation of $\mathcal{Q}$ is governed by the state machine depicted in Fig. 3.

A. Evolution of $\mathcal{Q}$

We now define the unitary evolution operators associated with the $\mathcal{Q}$ state transitions. In the equations below, subscripts on projectors denote the qubit(s) on which the projector acts, e.g., $|i\rangle\langle i|_k$ acts on qubit k, and unspecified qubits are understood to be operated on by an implicit identity operator, e.g., $|i\rangle\langle j|_k \otimes |l\rangle\langle m|_n$ is short hand for $|i\rangle\langle j|_k \otimes |l\rangle\langle m|_n \otimes \mathbb{1}_{\neq k,n}$ which acts on qubits k and n and leaves all other qubits unaffected. $|\psi\rangle_R$ denotes $\Pi_i |\psi(i)\rangle_{R(i)}$, where R is a multiple qubit register (e.g., D) with $R=\Pi_i R(i)$ and $\psi=\Pi_i \psi(i)$. Moreover, for notational simplicity in the rest of this paper, we define four primitive unitary operations, SWAP, DEC, INC, and NAND as follows:

(1) Swap contents of registers a and b,

$$\mathrm{SWAP}_{a,b} \overset{\mathrm{def}}{=} \sum_{i,j} |i\rangle\langle j|_a \otimes |j\rangle\langle i|_b. \tag{1}$$

(2) Decrement the contents of register a,

$$\mathrm{DEC}_a \overset{\mathrm{def}}{=} \sum_i |i-1\rangle\langle i|_a. \tag{2}$$

(3) Increment the contents of register a,

$$\mathrm{INC}_a \overset{\mathrm{def}}{=} \sum_i |i+1\rangle\langle i|_a. \tag{3}$$

(4) CNOT operation using qubit a as the control and qubit b as the data,

$$\mathrm{CNOT}_{a,b} \overset{\mathrm{def}}{=} (|00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 10| + |10\rangle\langle 11|)_{a,b}. \tag{4}$$

The operators, U_{IF}, U_{XD}, U_{EX}, U_{UD}, U_{UF}, U_{RI}, U_{UP}, and U_{CF}, which govern the $\mathcal{Q}$ state transitions, then, are defined as follows.

(1) Fetch next instruction at $M(P)$,

$$U_{\mathrm{IF}} \overset{\mathrm{def}}{=} \mathrm{DEC}_P^4 \cdot \mathrm{SWAP}_{x,P} \cdot \left(\prod_{i=3}^{0} \mathrm{INC}_x \cdot \mathrm{SWAP}_{M(x),I(i)} \right) \mathrm{SWAP}_{x,P}. \tag{5}$$

This operator fetches the next program instruction by "swapping" the next program instruction qubits on the memory tape with the contents of the I register. As stated earlier, because I is initialized to $|0\rangle$, the instruction slot on the memory tape becomes temporarily $|0\rangle$ while the instruction is being executed but is restored to its original state once the instruction has been executed. Note that P will be pointing back to the fetched instruction address after this operator is applied because the update of the program counter is deferred until U_{UP} is applied.

(2) Move tape head to $M(D)$,

$$U_{\mathrm{XD}} \overset{\mathrm{def}}{=} \mathrm{SWAP}_{x,D}. \tag{6}$$

U_{XD} points the memory tape head to the qubit addressed by the data register D.

(3) Execute instruction

$$\begin{aligned} U_{\mathrm{EX}} \overset{\mathrm{def}}{=} & |\mathrm{NOP}\rangle\langle\mathrm{NOP}|_I + |D \to 0\rangle\langle D \to 0|_I \otimes \mathrm{DEC}_x \\ & + |D+1\rangle\langle D+1|_I \otimes \mathrm{INC}_x + |D-1\rangle\langle D-1|_I \otimes \mathrm{DEC}_x \\ & + |\mathrm{H}\rangle\langle\mathrm{H}|_I \otimes \mathrm{H}_{M(x)} + |\mathrm{T}\rangle\langle\mathrm{T}|_I \otimes \mathrm{T}_{M(x)} \\ & + |\mathrm{SWAP}\rangle\langle\mathrm{SWAP}|_I \otimes \mathrm{SWAP}_{M(x),s} \\ & + |\mathrm{CNOT}\rangle\langle\mathrm{CNOT}|_I \otimes \mathrm{CNOT}_{s,M(x)} \\ & + |D \leftrightarrow P\rangle\langle D \leftrightarrow P|_I + |\mathrm{CLS}\rangle\langle\mathrm{CLS}|_I \\ & \otimes \mathrm{SWAP}_{x,H} \cdot \mathrm{DEC}_x \cdot \mathrm{SWAP}_{M(x),s} \cdot \mathrm{SWAP}_{x,H} \\ & + |h \to 1\rangle\langle h \to 1|_I \otimes (|1\rangle\langle 0| + |0\rangle\langle 1|)_h + \sum_{i=0}^{4} |\mathrm{R_i}\rangle\langle \mathrm{R_i}|_I. \end{aligned} \tag{7}$$

U_{EX} applies the appropriate transformation associated with the instruction being executed. The transformations associated with the NOP, $D+1$, $D-1$, H, T, SWAP, CNOT, and reserved instructions are self-evident but those associated with the $D \to 0$, $D \leftrightarrow P$, CLS, and $h \to 1$ instructions warrant some explanation.

The $D \to 0$ instruction works as follows. D is decremented by one by U_{EX} and P is left unchanged by U_{UP} [see Eq. (11)] until $D=0$. Leaving the program counter unchanged has the effect of keeping P pointing to the $D \to 0$ instruction such that it is refetched in the next iteration. Thus, $\mathcal{Q}$ continues to fetch and execute the same $D \to 0$ instruction until $D=0$. In other words, it will loop on the $D \to 0$ instruction, decrementing D until it reaches 0. Once $D=0$, P is incremented by 5 such that it points to the next instruction, thus completing the loop.

It is important to note that this scheme relies on the assumption that $D>0$ when the $D \to 0$ instruction is encountered. Therefore, the programmer must ensure that $D>0$ when $\mathcal{Q}$ fetches the $D \to 0$ instruction. This can be accomplished by preceding the $D \to 0$ instruction with a $D+1$ instruction since, in the absence of programming error, D will always be positive. If the programmer fails to meet this requirement $\mathcal{Q}$ could loop forever stepping through the negative portions of the memory tape.

The transformation associated with the $D \leftrightarrow P$ operation is the identity operation here because its execution is deferred until later. Deferring the actual swapping of the D and P register contents is necessary in order to keep the address of the $D \leftrightarrow P$ instruction unchanged so that we can restore the branch instruction back to its original slot on the memory tape and only then update the program counter to point to the next instruction in the program execution flow.

The CLS instruction first points the memory tape head to the address contained in the H register (the next slot on the negative portion of the memory tape that contains $|0\rangle$), swaps the contents of the s qubit with the contents of the memory tape slot ($|0\rangle$) thereby clearing s (but leaving the previous value of s on the memory tape making the operation reversible in principle), decrements H such that it points to the next $|0\rangle$ slot on the memory tape, and then points the memory tape head back to where it was.

The NOP instruction plays a key role in the $\mathcal{UQC}$ halting scheme. In our implementation, the halting scheme requires the halt instruction $|h \to 1\rangle$ to be followed by a $|\mathrm{NOP}\rangle$ instruction. In other words, the "true" halt instruction is effectively $|h \to 1\rangle|\mathrm{NOP}\rangle$ or $|11110000\rangle$ using the presently defined instruction encodings. This is such that after the halt qubit is set, $\mathcal{Q}$ will continue to fetch the next instruction following $|h \to 1\rangle$ which being $|\mathrm{NOP}\rangle$ will guarantee that $\mathcal{Q}$ loops forever doing nothing, thereby effectively halting program execution (but not quantum evolution). The fact that the encoding of the NOP instruction is $|0000\rangle$ is also intentional. This ensures that the contents of the memory tape remain unchanged after the halt qubit is set because swapping the instruction slot on the memory tape with the contents of the I register leaves the state of the memory tape unchanged. The halting scheme relies on all halt instructions in any given program being followed by a NOP instruction and stopping the program counter from changing when a NOP instruction is executed such that P will continue to point at the NOP instruction following the instruction that caused the halt qubit to be set.

The halting scheme is thus effectively a two step process: the first step is to set the halt qubit using the $h \to 1$ instruction to alert an external observer that the program has halted and the second step is to loop forever on the NOP instruction. In this sense, the NOP instruction is really a "loop forever" trap instruction. As such, the NOP instruction must only be used following a halt instruction. If it is inadvertently placed

TABLE II. F encodings.

Label	Encoding	Description
$\|\mathrm{LOOP}\rangle$	$\|00\rangle$	Loop $(P \to P)$
$\|\mathrm{NEXT}\rangle$	$\|01\rangle$	Next sequential instruction $(P \to P+5)$
$\|\mathrm{BR2D}\rangle$	$\|10\rangle$	Branch to D $(P \to D)$
$\|\mathrm{R}_0\rangle$	$\|11\rangle$	Unused

anywhere else in the program, program execution will halt but the halt qubit will not be set so the external observer will not know that the program has halted.

An improved halting scheme that does not require all instances of the halt instruction in a program to be followed by a NOP instruction may be possible and is an area for future investigation.

(4) Update contents of D register,

$$U_{\mathrm{UD}} \overset{\mathrm{def}}{=} \mathrm{SWAP}_{x,D}. \tag{8}$$

U_{UD} updates the D register with the results of executing the instruction since x will contain any changes to D after U_{EX} has been applied.

(5) Update control flow register with instruction flow information

$$\begin{aligned} U_{\mathrm{UF}} \overset{\mathrm{def}}{=} & [|D \to 0\rangle\langle D \to 0|_I \otimes (\mathbb{1} - |0\rangle\langle 0|)_D + |\mathrm{NOP}\rangle\langle \mathrm{NOP}|_I] \\ & + |D \leftrightarrow P\rangle\langle D \leftrightarrow P|_I \otimes |0\rangle\langle 0|_s \otimes \mathrm{INC}_F^2 \\ & + [\mathbb{1} - |D \to 0\rangle\langle D \to 0|_I \otimes (\mathbb{1} - |0\rangle\langle 0|)_D - |\mathrm{NOP}\rangle\langle \mathrm{NOP}|_I \\ & - |D \leftrightarrow P\rangle\langle D \leftrightarrow P|_I \otimes |0\rangle\langle 0|_s] \otimes \mathrm{INC}_F \end{aligned} \tag{9}$$

U_{UF} updates F whose value is later used to update P to point to the address of the next instruction to be executed. Note that F is initialized to $|0\rangle$ and the evolution of $\mathcal{UQC}$ is designed to ensure that $F=0$ when U_{UF} is applied (F is effectively "cleared" by U_{CF} by swapping its contents with the infinite supply of $|0\rangle$ slots on the negative portion of the memory tape as we describe later). As explained earlier, if the instruction is $D \to 0$ and $D \neq 0$ or if the instruction is NOP, P will be left unchanged to effectively loop on the instruction. If the instruction is $D \leftrightarrow P$ and $s=0$ then P will be swapped with D to effectively branch to D. Otherwise, P is set to point to the instruction following the instruction that was just executed (i.e., $P \to P+5$). The encodings of F are defined in Table II.

(6) Restore executed instruction back to the memory tape location from where it was fetched,

$$U_{\mathrm{RI}} \overset{\mathrm{def}}{=} U_{\mathrm{IF}}^{\dagger}. \tag{10}$$

U_{RI} restores the instruction that was just executed back to its original slot on the memory tape. Recall that the P update has been deferred and will be controlled by the state of the F register. Thus, the only operator that has affected P thus far has been U_{IF} so $U_{\mathrm{IF}}^{\dagger}$ suffices to undo the fetch. In essence, F is a temporary place holder to store the information necessary to determine the next instruction location after restoring the instruction back to the memory tape and hence losing knowledge of how to update P otherwise.

(7) Update program counter to the address of the next instruction to be executed,

$$\begin{aligned} U_{\mathrm{UP}} \overset{\mathrm{def}}{=} & |\mathrm{LOOP}\rangle\langle \mathrm{LOOP}|_F + |\mathrm{NEXT}\rangle\langle \mathrm{NEXT}|_F \otimes \mathrm{INC}_P^5 \\ & + |\mathrm{BR2D}\rangle\langle \mathrm{BR2D}|_F \otimes \mathrm{SWAP}_{D,P} + |\mathrm{R}_0\rangle\langle \mathrm{R}_0|_F. \end{aligned} \tag{11}$$

U_{UP} updates P to the address of the next instruction to be executed according to the state of F.

(8) Clear flow control register such that it can be used again in the next cycle,

$$U_{\mathrm{CF}} \overset{\mathrm{def}}{=} \mathrm{SWAP}_{x,H}\left(\prod_{i=1}^{0} \mathrm{DEC}_x \cdot \mathrm{SWAP}_{M(x),F(i)}\right)\mathrm{SWAP}_{x,H}. \tag{12}$$

U_{CF} first swaps the contents of the x and H registers. The H register contains the address of the next slot on the negative portion of the tape that contains $|00\rangle$. These "0" slots are used to clear the F register back to $|0\rangle$ each cycle. Since the sequence of F values effectively contains the information about the program execution flow, in essence the negative portion of the tape contains the "history" of instructions that $\mathcal{UQC}$ has executed and is a side effect of the need for all $\mathcal{UQC}$ programs to be reversible.

In other words, the negative portion of M is used to store the ancillary garbage data that would be required to reverse the operation of the program. The number of $|0\rangle$ slots required for any given program is equal to the number of instructions that are executed by the program. U_{CF} clears F by swapping its contents with the contents of the next $|0\rangle$ slot on the negative portion of the tape. After application of U_{CF}, H points to the next $|0\rangle$ slot on the tape and the previous F value is contained on the slot to the right of the first $|0\rangle$ slot on the negative portion of the tape. At this point, $\mathcal{Q}$ has completed processing the instruction and is ready to fetch the next instruction in the execution flow.

These operators are all readily verified to be unitary and to have the desired effects of implementing the state machine shown in Fig. 3. The overall evolution of $\mathcal{UQC}$, then, is governed by the unitary operator,

$$U = U_{\mathrm{CF}}U_{\mathrm{UP}}U_{\mathrm{RI}}U_{\mathrm{UF}}U_{\mathrm{UD}}U_{\mathrm{EX}}U_{\mathrm{XD}}U_{\mathrm{IF}}. \tag{13}$$

Unlike Deutsch's original UQTM, the memory tape (or tape head) of $\mathcal{UQC}$ is not restricted to move at most one position to the left or to the right $(x \to x \pm 1)$ in any given step. This is most obvious in the case of the branch instruction where the tape head will jump by an arbitrarily large amount in a single step. However, the evolution of $\mathcal{UQC}$ is still unitary and hence physically possible in principle.

VI. SOME PRIMITIVE PROGRAMS

In this section we describe a set of primitive programs or operations to demonstrate the universal nature of $\mathcal{UQC}$. These routines serve as building blocks for devising and analyzing more complicated and useful programs.

The first set of primitive programs, $\{|D_{+i}\rangle, |D_{-i}\rangle, |D_i\rangle, |S_{i,s}\rangle, |S_{i,j}\rangle, |B_i\rangle\}$, that we define perform basic functions to manipulate the data address register, swap qubits, and conditionally branch to an arbitrary address. The superscripts on the programs denote the operation performed by the program and the subscripts indicate the qubits on which the program operates. For notational simplicity, $|P_h\rangle$ denotes the program that causes $\mathcal{UQC}$ to halt, i.e., $|P_h\rangle \overset{\text{def}}{=} |h \to 1\rangle|\text{NOP}\rangle$.

(1) $|D_{+i}\rangle$: Increment D by i,

$$|D_{+i}\rangle \overset{\text{def}}{=} \begin{cases} \prod_{k=1}^{i} |D+1\rangle, & \text{if } i \geq 1 \\ \mathbb{1}, & \text{otherwise.} \end{cases} \tag{14}$$

(2) $|D_{-i}\rangle$: Decrement D by i,

$$|D_{-i}\rangle \overset{\text{def}}{=} \begin{cases} \prod_{k=1}^{i} |D-1\rangle, & \text{if } i \geq 1 \\ \mathbb{1}, & \text{otherwise.} \end{cases} \tag{15}$$

(3) $|D_i\rangle$: Set D to i, $i>0$,

$$|D_i\rangle \overset{\text{def}}{=} |D+1\rangle|D \to 0\rangle|D_{+i}\rangle. \tag{16}$$

Recall from the discussion of U_{EX} that we are preceding the $D \to 0$ instruction with a $D+1$ instruction to ensure that $D>0$ when the $D \to 0$ instruction is executed.

(4) $|S_{i,s}\rangle$: Swap data qubits $D(i)$ and s,

$$|S_{i,s}\rangle \overset{\text{def}}{=} |D_{5i-1}\rangle|\text{SWAP}\rangle. \tag{17}$$

(5) $|S_{i,j}\rangle$: Swap data qubits $D(i)$ and $D(j)$,

$$|S_{i,j}\rangle \overset{\text{def}}{=} |S_{5i-1,s}\rangle|S_{5j-1,s}\rangle|S_{5i-1,s}\rangle. \tag{18}$$

(6) Branch to the ith instruction [i.e., instruction at $M(5(i-1))$], where $i \in \mathbb{Z}^+$,

$$|B_i\rangle \overset{\text{def}}{=} |D_{5(i-1)}\rangle|D \leftrightarrow P\rangle. \tag{19}$$

Note that, as defined, this instruction will have no effect unless $|s\rangle=|0\rangle$ so this operation is only useful following nontrivial operations on the $|s\rangle$ qubit.

Next we describe a set of programs, $\{|P_i^{\text{H}}\rangle, |P_{i,j}^{\text{H}}\rangle, |P_i^{\text{T}}\rangle, |P_{i,j}^{\text{C}}\rangle\}$, to apply the H, T, and CNOT operations on arbitrary qubits i and j on the memory tape, where i and $j \in \mathbb{Z}$. These comprise a universal set of unitary operations from which any arbitrary unitary operation can be constructed.

(1) $|P_i^{\text{H}}\rangle$: Apply H to data qubit $D(i)$,

$$|P_i^{\text{H}}\rangle \overset{\text{def}}{=} |D_{5i-1}\rangle|\text{H}\rangle. \tag{20}$$

(2) $|P_{i,j}^{\text{H}}\rangle$: Apply H to data qubits $D(i:j)$, where $i \geq j$,

$$|P_{i,j}^{\text{H}}\rangle \overset{\text{def}}{=} \prod_{k=j}^{i} |P_k^{\text{H}}\rangle. \tag{21}$$

One could implement this program using a loop but that would require first implementing binary addition of M qubits. Binary addition is possible because one can implement a binary adder such as a Carry Lookahead Adder [14] using the NAND program that we define later in this section. However, since we are only interested in a polynomial order (in the number of qubits) multiple qubit Hadamard transformation program, we define $|P_{i,j}^{\text{H}}\rangle$ as a sequential "unrolled" loop program.

(3) $|P_i^{\text{T}}\rangle$: Apply T to data qubit $D(i)$,

$$|P_i^{\text{T}}\rangle \overset{\text{def}}{=} |D_{5i-1}\rangle|\text{T}\rangle. \tag{22}$$

(4) $|P_{i,j}^{\text{C}}\rangle$: Apply CNOT to data qubits $D(i)$ and $D(j)$ with $D(i)$ as the control qubit,

$$|P_{i,j}^{\text{C}}\rangle \overset{\text{def}}{=} |S_{i,s}\rangle|D_{5j-1}\rangle|\text{CNOT}\rangle|S_{i,s}\rangle. \tag{23}$$

Using the sets of primitive programs defined above, we can now define the set of programs, $\{|P_i^{\text{X}}\rangle, |P_i^{\text{S}}\rangle, |P_i^{\text{T}^\dagger}\rangle\}$, that apply the Pauli X, Phase (S), and $\text{T}^\dagger$ operations on data qubit $i \in \mathbb{Z}^+$. These operations are often used in quantum algorithms so it is useful to identify the programs that implement them. A constant subscript on a program denotes that some suitable qubit on the memory tape has been prepared with the appropriate value. For example, $|P_1\rangle$ is shorthand for $|P_k\rangle$ where $M(k)$, for some suitable k, has been prepared with the value $|1\rangle$.

(1) $|P_i^{\text{X}}\rangle$: Apply σ_x to qubit $M(i)$,

$$|P_i^{\text{X}}\rangle = |P_{1,i}^{\text{C}}\rangle,$$

$$|P_i^{\text{X}}\rangle : |1\rangle_j|\psi\rangle_i \to |1\rangle_j|1 \oplus \psi\rangle_i = |1\rangle_j|\bar{\psi}\rangle_i = |1\rangle_j\sigma_x|\psi\rangle_i. \tag{24}$$

(2) $|P_i^{\text{S}}\rangle$: Apply phase (S) to qubit $M(i)$.

Noting that $\text{S}=\text{T}^2$, the following program implements the phase operation.

$$|P_i^{\text{S}}\rangle = |P_i^{\text{T}}\rangle|P_i^{\text{T}}\rangle$$

$$|P_i^{\text{S}}\rangle : |\psi\rangle_i \to \text{T}^2|\psi\rangle_i = \text{S}|\psi\rangle_i. \tag{25}$$

(3) $|P_i^{\text{T}^\dagger}\rangle$: Apply $\text{T}^\dagger$ (reverse T) to qubit $M(i)$,

This operation is used to define the Toffoli operation in a later section so we define it here. Noting that $\text{S}=\text{T}^2$ and that $S^4=\mathbb{1}$, $\text{T}^\dagger=\text{T}^\dagger\text{S}^4=\text{T}^\dagger\text{T}^2\text{S}^3=\text{TS}^3$, the following program implements the $\text{T}^\dagger$ operation:

$$|P_i^{\text{T}^\dagger}\rangle = |P_i^{\text{S}}\rangle|P_i^{\text{S}}\rangle|P_i^{\text{S}}\rangle|P_i^{\text{T}}\rangle. \tag{26}$$

Although not specifically shown here, other useful quantum gates such as σ_y, σ_z, entanglement gate, etc. can be similarly implemented. These enable us to implement any algorithm from the quantum gate array framework on $\mathcal{UQC}$ by appropriate combinations of the programs we have just defined and adding $|P_h\rangle$ as the last step in the combined program to halt $\mathcal{UQC}$ upon completion. Since the quantum gate array framework is universal (see [4], Chap. 4, for example), this means that $\mathcal{UQC}$ is also quantum computationally universal with the additional advantage that $\mathcal{UQC}$ provides a fixed and programmable machine to implement the

algorithms unlike the quantum gate array framework.

The two-bit NAND operation is universal for classical computation. That is, the NAND operation can be used to implement any Boolean function. Hence it is useful to define a program that emulates the NAND operation on two qubits as this could be used as the basis for emulating classical functions on $\mathcal{UQC}$. For this purpose, we first define a program that implements the Toffoli operation which itself is a universal classical gate [4]. The Toffoli program, $|P^{\mathrm{Toff}}_{i,j,k}\rangle$, applies the Toffoli operation to qubits $D(i)$, $D(j)$, and $D(k)$, where $D(i)$ and $D(j)$ are the control qubits and $D(k)$ is the target qubit.

Armed with the Toffoli program, implementing a program that takes the NAND of qubits $D(i)$ and $D(j)$ and storing the result in qubit $D(c)$ is a simple matter of executing the program $|P^{\mathrm{NAND}}_{i,j,c}\rangle = |P^{\mathrm{Toff}}_{i,j,1}\rangle$,

$$|P^{\mathrm{Toff}}_{i,j,k}\rangle = |P^{\mathrm{H}}_{k}\rangle|P^{\mathrm{C}}_{j,k}\rangle|P^{\mathrm{T}^{\dagger}}_{k}\rangle|P^{\mathrm{C}}_{i,k}\rangle|P^{\mathrm{T}}_{k}\rangle|P^{\mathrm{C}}_{j,k}\rangle|P^{\mathrm{T}^{\dagger}}_{k}\rangle|P^{\mathrm{C}}_{i,k}\rangle|P^{\mathrm{T}}_{k}\rangle|P^{\mathrm{T}^{\dagger}}_{j}\rangle|P^{\mathrm{C}}_{i,j}\rangle \times |P^{\mathrm{H}}_{k}\rangle|P^{\mathrm{T}^{\dagger}}_{j}\rangle|P^{\mathrm{C}}_{i,j}\rangle|P^{\mathrm{S}}_{j}\rangle|P^{\mathrm{T}}_{i}\rangle. \quad (27)$$

The ability to perform a two-qubit NAND operation gives $\mathcal{UQC}$ the ability to compute any classically computable function thus demonstrating that it can emulate a classical universal Turing machine. This is in addition to being a universal quantum computer since it can also implement the set of universal quantum operations on arbitrary qubits on its memory tape as shown earlier. $\mathcal{UQC}$ can compute any classically computable function, it can compute any quantum computable function, and it is programmable. In short, $\mathcal{UQC}$ is computationally universal.

VII. PROGRAM CONCATENATION SCHEME

In the process of defining the primitive programs in the preceding section, we have implicitly used program concatenation whereby we sequentially combined separate programs to create larger programs. Strictly speaking, the programs that we have thus far defined are really subroutines since complete programs must include the halting program, $|P_h\rangle$, in order to signal program completion. However, it is readily seen that all of the primitive subroutines can be converted into full-fledged programs by adding $|P_h\rangle$ as the last instruction.

Sequential programs (programs without branch instructions) can thus be concatenated by simply removing the last $|P_h\rangle$ step from each constituent program, concatenating the resulting subroutines, and appending on $|P_h\rangle$ at the end. Suppose that we have two sequential programs, $|P_A\rangle$ and $|P_B\rangle$, that we wish to concatenate to create a program $|P_{AB}\rangle$ whose effect is to execute $|P_B\rangle$ followed by $|P_A\rangle$. Since $|P_A\rangle$ and $|P_B\rangle$ are sequential, this means that $|P_h\rangle$ is the last step in each program. That is, $|P_A\rangle = |P_{A'}\rangle|P_h\rangle$ and $|P_B\rangle = |P_{B'}\rangle|P_h\rangle$. Thus, to achieve the effect of running $|P_A\rangle$ followed by $|P_B\rangle$, we simply construct the program $P_{AB} \overset{\mathrm{def}}{=} |P_{A'}\rangle|P_{B'}\rangle|P_h\rangle$.

The situation is quite different for branching programs. In general, without complete knowledge of the operations of the programs to be concatenated, it is not possible to concatenate them in the strictest sense of joining the individual programs into a single larger program. This is not a limitation of $\mathcal{UQC}$ but of any computer, be it classical or quantum. The problem is that the branch destinations in a branching program can be data dependent and the branching address may also be manipulated as data. Therefore, it is not sufficient to add an appropriate offset (the number of instructions of preceding concatenated programs) to all branch instructions because this would have the adverse effect of potentially adding an offset to the manipulated data and hence altering the intended computation results.

The solution, of course, is to first run $|P_A\rangle$, wait for it to complete, replace $|P_A\rangle$ with $|P_B\rangle$, reset the program counter register to 0, leave all other internal registers and memory tape qubits unchanged, and resume execution to run $|P_B\rangle$. However, strictly speaking, this is not program concatenation *per se* because while the overall operation has the effect of running $|P_A\rangle$ followed by $|P_B\rangle$, the program that is run is not $|P_A\rangle|P_B\rangle$. There is the intermediate step of replacing $|P_A\rangle$ with $|P_B\rangle$ and restarting execution, which, strictly speaking, are not program operations. In the context of $\mathcal{UQC}$ this could be achieved by initializing M with $|P_A\rangle$, running $|P_A\rangle$ and once the halt qubit is measured as $|1\rangle$, replacing the program portion of M with $|P_B\rangle$, setting the program counter register to 0 while leaving all other $\mathcal{UQC}$ registers and memory qubits unchanged, and clearing the halt qubit to resume execution of $|P_B\rangle$ with the results of the preceding program(s). This scheme, of course, works not only for branching programs but also for sequential programs.

Formally, our $\mathcal{UQC}$ program concatenation operator, $\Pi^{(n)}$, is defined as

$$\Pi^{(n)} \overset{\mathrm{def}}{=} \left(\sum_{i=1}^{n-1} |P_{i+1}\rangle\langle P_i|_{M(P)} \otimes |0\rangle\langle 1|_h \otimes \mathrm{SWAP}_{P,P'(i)} \otimes |1\rangle\langle 0|_{S(i+1)}\right) + |P_1\rangle\langle 0|_{M(P)} \otimes |0\rangle\langle 0|_h \otimes |1\rangle\langle 0|_{S(1)} + \left(\sum_{i=1}^{n-1} |P_i\rangle\langle P_{i+1}|_{M(P)} \otimes |1\rangle\langle 0|_h \otimes |1\rangle\langle 0|_{S(i)}\right) + |P_n\rangle\langle P_n|_{M(P)} \otimes |1\rangle\langle 1|_h \otimes |1\rangle\langle 0|_{S(n)} + \sum_{i=1}^{n} |P_i\rangle\langle P_i|_{M(P)} \otimes |0\rangle\langle 1|_{S(i)} + \sum_{i=n+1}^{\infty} |P_i\rangle\langle P_i|_{M(P)}, \quad (28)$$

where n denotes the number of programs to be concatenated, $|P_i\rangle$ denotes the ith program in the concatenation sequence (we are assuming that the programs have been enumerated such that the first n programs are the ones that we wish to concatenate), $M(P)$ denotes the program qubits portion of M, h denotes the halt qubit, P denotes the program counter register, $P'(i)$ denotes the ith ancillary program counter register, and $S(i)$ is the ith flag denoting that program i has been swapped. Note that some suitable finite unused subset of M can be used for $P'(i)$ and $S(i)$ since these are initialized to 0 and are only used once in the program concatenation operation. The P' and S' arrays of qubits are required to save intermediate states during program swaps to ensure unitarity.

In order to concatenate n given programs, then, we simply modify the overall $\mathcal{UQC}$ evolution operator to

$$U \overset{\text{def}}{=} \Pi^{(n)} U_{\mathrm{CF}} U_{\mathrm{UP}} U_{\mathrm{RI}} U_{\mathrm{UF}} U_{\mathrm{UD}} U_{\mathrm{EX}} U_{\mathrm{XD}} U_{\mathrm{IF}}. \tag{29}$$

U then has the net effect of running each program until it halts, swapping each completed program with the next program in the concatenation sequence, swapping the program counter register with 0, flipping the halt qubit (and hence starting execution of the swapped program), and leaving the final result on the tape when the last program, $|P_n\rangle$, halts. Even if the individual programs are known to halt, the concatenated program will not necessarily halt because, in general, the input data to the individual programs will change when run as part of a concatenated program. Hence, whether or not a concatenated program will halt is independent of whether or not its constituent programs halt.

The famous Turing halting problem is only relevant in the context of executing programs that can branch. Nonbranching finite programs, by construction, will always halt so the halting problem is a moot point in that case. This raises a question about Deutsch's UQTM. Deutsch did not explicitly consider branching in his original UQTM proposal and thus it is unclear whether or not his program concatenation scheme rested on the assumption that UQTM programs were nonbranching. If UQTM programs could involve branching, then without guaranteed halting of the concatenated programs, the validity of Deutsch's program concatenation scheme is problematic. Deutsch's description of his program concatenation scheme suggests that it was an "appending" scheme rather than a "swapping" scheme as we have defined.

There is still one problem with the program concatenation scheme. As currently defined, the halt qubit will be flipped several times during the course of executing a concatenation of programs (assuming that each constituent program halts, of course). Thus, it may appear that there is no way for an external observer (or classical computer) to distinguish between the intermediate and final states of the halt qubit. However, this does not pose a problem so long as the measurement of the halt qubit does not affect the result that we will ultimately measure on the memory tape in which case we simply measure the halt qubit, wait the time associated with program swap operations (i.e., $|P_{i+1}\rangle\langle P_i|$) to be completed, and measure the halt qubit again. If the halt qubit was in an intermediate set state, we will then find it cleared. If, on the other hand, the halt qubit was in its final set state, then we will find it still set and we can then measure the memory tape to find the result. Thus, periodic measurements of the halt qubit suffice to identify whether the concatenated program has halted. The question, then, is whether periodic measurements of the halt qubit affect the final measurement of the memory tape. Ozawa [10] has already proven that periodic measurement of the halt qubit does not spoil the result of the computation (i.e., the final measurement of the memory tape contents). That is, the probability of finding the memory tape in state M_i after N iterations of U with periodic measurements (monitoring) of the halt qubit and the probability of finding the memory tape in the state M_i after N iterations of U without periodic measurements of the halt qubit (i.e., one single measurement of M after N iterations of U) are identical. Thus, periodic measurements of the halt qubit do not spoil the intermediate computation as Myers argued.

Therefore, we see that concatenation of $\mathcal{UQC}$ programs works in the same way as concatenation of classical computer programs. While the halting question for the resultant program still remains just as it does for classical computers, a valid unitary $\mathcal{UQC}$ program concatenation scheme exists. The programs to be concatenated are sequentially executed without changing the state of internal registers except for the program counter. Not surprisingly, the concatenation scheme is analogous to the classical case.

VIII. CONCLUSION

The quantum computer we have defined is universal in the sense that, under the control of quantum programs, it can first emulate any classical Turing machine by being able to compute the NAND function and second can approximate any unitary operation to any desired accuracy by being able to apply the set of {H,CNOT,T} operations on a specified set of qubits. The machine also supports conditional branching and hence conditional execution, a feature that is not directly possible in the quantum gate array circuit framework. The defined halting scheme works in a way that prevents changes to the memory tape once the program has halted thus satisfying Ozawa's proof requirement and allowing for a valid program concatenation scheme. Because of its universality, $\mathcal{UQC}$ serves as a prototypical model for general-purpose programmable quantum computation and should find uses in the development and analysis of quantum algorithms and complexity. Work in progress using the $\mathcal{UQC}$ includes a demonstration of how to implement oracle based algorithms such as the Grover search algorithm.

[1] *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, edited by S. Goldwasser (IEEE Computer Society, Los Alamitos, CA, 1994).
[2] T. Kieu, Int. J. Theor. Phys. **42**, 1461 (2003).
[3] D. Deutsch, Proc. R. Soc. Lond. **425**, 73 (1989).
[4] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, England, 2000).
[5] L. K. Grover, Phys. Rev. Lett. **79**, 325 (1997).
[6] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, e-print arXiv:quant-ph/0001106.
[7] D. Deutsch, Proc. R. Soc. Lond. **400**, 97 (1985).
[8] J. M. Myers, Phys. Rev. Lett. **78**, 1823 (1997).
[9] Y. Shi, Phys. Lett. A **293**, 277 (2002).
[10] M. Ozawa, Phys. Rev. Lett. **80**, 631 (1998).
[11] N. S. Yanofski and M. A. Mannucci, *Quantum Computing for Computer Scientists* (Cambridge University Press, Cambridge, England, 2008).
[12] E. Bernstein and U. Vazirani, SIAM J. Comput. **26**, 1411 (1997).
[13] S. Iriyama, T. Miyadera, and M. Ohya, Phys. Lett. A **372**, 5120 (2008).
[14] S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers* (Holt, Rinehart and Winston, New York, 1982).

Appendix C

Published Work

INTERFACING EXTERNAL QUANTUM DEVICES TO A UNIVERSAL QUANTUM COMPUTER

Antonio A. Lagana, M. A. Lohe, Lorenz von Smekal

Department of Physics, University of Adelaide
South Australia 5005, Australia

PLoS ONE 6(12): e29417. doi:10.1371/journal.pone.0029417 (2011)

STATEMENT OF AUTHORSHIP

INTERFACING EXTERNAL QUANTUM DEVICES TO A UNIVERSAL QUANTUM COMPUTER

PLoS ONE 6(12): e29417. doi:10.1371/journal.pone.0029417 (2011)

ANTONIO A. LAGANA (Candidate)

Devised scheme to interface universal quantum computer to external quantum devices. Developed universal quantum computer programs to realize well known oracle based algorithms using external black-box quantum oracle devices.

I hereby certify that the statement of contribution is accurate.

Signed Date. 24/3/2010

M. A. LOHE

Provided expertise in quantum mechanics, reviewed research, highlighted areas for deeper explanation and focus, provided constructive criticism and better ways to present the research work.

I hereby certify that the statement of contribution is accurate and I give permission for the inclusion of the paper in the thesis.

Signed Date. 24/3/2010

LORENZ VON SMEKAL

Provided expertise in quantum mechanics, reviewed research, highlighted areas for deeper explanation and focus, provided constructive criticism and better ways to present the research work.

I hereby certify that the statement of contribution is accurate and I give permission for the inclusion of the paper in the thesis.

Signed Date. 23.3.2010

OPEN ACCESS Freely available online

PLoS one

Interfacing External Quantum Devices to a Universal Quantum Computer

Antonio A. Lagana[1]*, Max A. Lohe[2], Lorenz von Smekal[3]

1 School of Chemistry and Physics, University of Adelaide, Adelaide, South Australia, Australia, **2** School of Chemistry and Physics, University of Adelaide, Adelaide, South Australia, Australia, **3** Institut für Kernphysik, Technische Universität Darmstadt, Darmstadt, Germany

Abstract

We present a scheme to use external quantum devices using the universal quantum computer previously constructed. We thereby show how the universal quantum computer can utilize networked quantum information resources to carry out local computations. Such information may come from specialized quantum devices or even from remote universal quantum computers. We show how to accomplish this by devising universal quantum computer programs that implement well known oracle based quantum algorithms, namely the Deutsch, Deutsch-Jozsa, and the Grover algorithms using external black-box quantum oracle devices. In the process, we demonstrate a method to map existing quantum algorithms onto the universal quantum computer.

Citation: Lagana AA, Lohe MA, von Smekal L (2011) Interfacing External Quantum Devices to a Universal Quantum Computer. PLoS ONE 6(12): e29417. doi:10.1371/journal.pone.0029417

Editor: Gerardo Adesso, University of Nottingham, United Kingdom

Received November 1, 2011; **Accepted** November 28, 2011; **Published** December 28, 2011

Funding: The authors have no funding or support to report.

Competing Interests: The authors have declared that no competing interests exist.

* E-mail: antonio.lagana@adelaide.edu.au

Introduction

Quantum networks which connect quantum systems and can transmit quantum information have been extensively discussed [1]. Quantum connectivity provides a means of overcoming size-scaling and error-correction problems, and has significant advantages over classical connectivity. Furthermore, networks of quantum computers have also been proposed [2] where information can be exchanged between nodes via quantum and classical channels. A general question arises as to whether and how such quantum computers can communicate and exchange information. In the simplest case a quantum computer may download data sets from other nodes over the quantum network, but in more complex cases use the network to call subroutines, or concatenate programs from other quantum computers.

It is well known that classical principles do not necessarily apply in the realm of quantum mechanics. The no-cloning theorem (see [3] for example) is a well-known example of this. In the field of quantum computing, the ability to halt a programmable quantum computer was such an example. The original Universal Quantum Turing Machine proposal [4] made the tacit assumption that a quantum turing machine could be halted in a classical manner. This turned out to be problematic (see [5] for a discussion of the issues associated with the original proposal) due to properties of quantum mechanics. Thus, it is imperative to formally show whether a classical solution or property is applicable (or even relevant) in the realm of quantum mechanics. Assuming that a classical solution to a problem directly applies to a quantum mechanical system is prone to run into potential complications.

We address here the question of how a universal quantum computer can access an external oracle, which may be regarded as a "black box" quantum device, possibly over a quantum network but in any case as a separate and external quantum system to the universal quantum computer itself. In fact, the oracle may be a program running on a remote universal quantum computer. It should be noted that this is a different problem from that of implementing an oracle "program" on a universal quantum computer. This is of course possible by virtue of the fact that the computer is universal. Hence, if such a program exists, it can be implemented and executed on a universal quantum computer. Strictly speaking, however, the ability to utilize external quantum devices over a network connection is a different problem because such devices are external to the universal quantum computer itself.

Classically, the ability to access devices on a network is a well-known problem with well-known solutions. However, as stated earlier, we cannot assume that this is necessarily the case for a quantum computer accessing quantum devices on a quantum network. Our aim is to explicitly show that accessing external quantum devices with a universal quantum computer is indeed possible by devising universal quantum computer programs that implement well-known oracle based quantum algorithms, namely the Deutsch, Deutsch-Jozsa, and the Grover algorithms using external black-box quantum oracle devices.

In [5] we constructed a programmable universal quantum computer UQC that is universal in the sense that it can emulate any classical Turing machine and can approximate any unitary operation to any desired accuracy. It is programmable in the sense that the machine's operations are specified using a sequence of instructions in the same way as for classical computers. UQC also supports conditional branching and hence conditional execution, a feature that is not directly possible in the quantum gate array circuit framework. Moreover, UQC uses a halting scheme that allows for valid program concatenation, thus resolving issues with the original Universal Quantum Turing Machine (UQTM) proposed by Deutsch [4].

In order to use information from a quantum network in *UQC* programs, we need to devise a means of enabling *UQC* programs to access such remote information and use that information for local computations. We assume that remote quantum nodes exist and treat them as black boxes without any assumptions as to their internal structure or operational details. Without loss of generality, we assume that such devices accept a finite number of input qubits and generate a finite number of output qubits. The input and output qubits may be shared, which is the case if the remote device functions in such a way as to alter the input qubits based on its function. We also assume, without loss of generality, that quantum network nodes have an "enable" qubit, $|\mathrm{en}\rangle$, that controls when an access is to begin, in order to let the device know when the input data has been prepared and is valid. We further assume, without loss of generality, that the nodes of the network generate their output data in less time than the time associated with a single iteration of *UQC*. If the query time were longer than a single iteration of *UQC* or were data-dependent, one could simply write the *UQC* program to wait for the appropriate number of cycles before using the result of the network access. Alternatively, the nodes could provide an "access completed" status flag qubit such that the *UQC* program could poll this status flag qubit before using the result of a network access.

Results

Recall from [5] that *UQC* consists of a memory tape M with an infinite number of qubits, of which only a finite portion is ever used, and a processor that contains observables that play the roles of several registers, including a data register D, a program counter register P, a scratch qubit s, and the halt qubit h. The processor executes programs stored on the memory tape using data that is also stored on the memory tape. A program of *UQC* consists of a sequence of qubits whose states encode instructions of the instruction set defined in [5] and reproduced in Table 1 at the end of this paper.

The single qubit operations H and T act on the qubit at tape location $M(D)$, denoted $|M\rangle_D$, and the two qubit operations SWAP and NAND act on $|M\rangle_D$ and the scratch qubit $|s\rangle$, the latter being used as the control qubit for the NAND operation.

The instruction set includes a set of operations that can approximate any unitary operation to any desired accuracy. Thus, it is quantum computationally universal. In [5] we constructed a *UQC* program that can compute the NAND function, thereby showing that the machine can compute any classically computable function. Because of *UQC*'s universality, any algorithm that can be implemented in the quantum gate array framework can be mapped to an equivalent *UQC* program by virtue of the fact that gate array circuits can be decomposed into circuits of gates with the same universal set of unitary operations {H,T,CNOT} that are implemented in the *UQC* instruction set. Each of the qubits in a quantum circuit (i.e. lines connecting gates) can be mapped to a suitable memory tape data qubit and each of the unitary operations (i.e. quantum gates) can be mapped to a suitable *UQC* subroutine. It is possible therefore to map quantum gate array implementations of algorithms such as the quantum Fourier transform, quantum phase estimation, quantum order finding, quantum factoring discussed in [6] (Chapter 5) onto *UQC*.

Table 1. *UQC* Instruction Set.

Label	Encoding	Description
$\|\mathrm{NOP}\rangle$	$\|0000\rangle$	No operation
$\|D\rightarrow 0\rangle$	$\|0001\rangle$	$D\rightarrow 0$
$\|D+1\rangle$	$\|0010\rangle$	$D\rightarrow D+1$
$\|D-1\rangle$	$\|0011\rangle$	$D\rightarrow D-1$
$\|\mathrm{H}\rangle$	$\|0100\rangle$	Apply Hadamard operation to $\|M\rangle_D$
$\|\mathrm{T}\rangle$	$\|0101\rangle$	Apply $\pi/8$ operation to $\|M\rangle_D$
$\|\mathrm{SWAP}\rangle$	$\|0110\rangle$	$\|M\rangle_D \leftrightarrow \|s\rangle$
$\|\mathrm{CNOT}\rangle$	$\|0111\rangle$	CNOT of $\|M\rangle_D$ and $\|s\rangle$ ($\|s\rangle$: control)
$\|D\leftrightarrow P\rangle$	$\|1000\rangle$	$\|D\rangle \leftrightarrow \|P\rangle$ (branch) iff $s=0$
$\|\mathrm{CLS}\rangle$	$\|1001\rangle$	Clear s
$\|h\rightarrow 1\rangle$	$\|1111\rangle$	$\|h\rangle \rightarrow \|1\rangle$ (set halt qubit)

doi:10.1371/journal.pone.0029417.t001

Accessing Networked Quantum Resources With *UQC*

Modifying *UQC* to use networked quantum devices, then, is a matter of connecting the qubits comprising the interface (input, output, enable, and optionally access complete) qubits of those devices to a finite subset of the data portion of M, which is the quantum analog of a classical computer's memory-mapped I/O and allows *UQC* programs to access remote devices using the M qubits that are connected to those devices. The *UQC* programs prepare the appropriate input data qubits, set the corresponding access enable qubits to perform an access, and utilize the corresponding output data qubits of M. It should be noted that a remote quantum device could be another instance of *UQC* which would enable distributed quantum computing. However, the scheme to access data from remote devices, be they simple devices or full-fledged quantum computers, would work in the same way.

Primitive Programs

In [5] we defined several primitive programs and subroutines that serve as building blocks for devising and analyzing more complicated and useful programs. We reproduce here only those that we specifically require for constructing the algorithms that are the focus of this work. By considering the quantum gate array framework implementations of the algorithms, we identify that we need programs that perform the operations H, σ_x, and CNOT. We also need to swap qubits for several operations such as enabling or disabling the remote networked quantum device, and the ability to address individual qubits on the memory tape to perform operations on them. Finally, we need a primitive program to halt the overall program.

In the equations that follow, superscripts on programs denote the operation specified by the program and subscripts indicate the qubits on which the program specifies the processor to operate upon. For notational simplicity, $|P_h\rangle$ denotes the program that halts *UQC*, i.e. $|P_h\rangle \stackrel{\mathrm{def}}{=} |h\rightarrow 1\rangle|\mathrm{NOP}\rangle$.

The first set of primitive programs, $\{|D_{+i}\rangle, |D_i\rangle, |S_{i,s}\rangle, |S_{i,j}\rangle\}$, is a subset of those defined in [5]:

1. $|D_{+i}\rangle$: Increment D by i,

$$|D_{+i}\rangle \stackrel{\mathrm{def}}{=} \begin{cases} \Pi_{k=1}^{i}|D+1\rangle & \text{if } i \geq 1, \\ \mathbb{I} & \text{otherwise.} \end{cases} \quad (1)$$

2. $|D_i\rangle$: Set D to i, $i>0$,

$$|D_i\rangle \stackrel{\mathrm{def}}{=} |D+1\rangle|D\rightarrow 0\rangle|D_{+i}\rangle. \quad (2)$$

Recall from the discussion of U_{EX} in [5], that we precede the $D\to 0$ instruction with a $D+1$ instruction to ensure that $D>0$ when the $D\to 0$ instruction is executed.

3. $|S_{i,s}\rangle$: Swap data qubits $D(i)$ and s,

$$|S_{i,s}\rangle \stackrel{\mathrm{def}}{=} |D_{5i-1}\rangle|\mathrm{SWAP}\rangle. \tag{3}$$

4. $|S_{i,j}\rangle$: Swap data qubits $D(i)$ and $D(j)$,

$$|S_{i,j}\rangle \stackrel{\mathrm{def}}{=} |S_{5i-1,s}\rangle|S_{5j-1,s}\rangle|S_{5i-1,s}\rangle. \tag{4}$$

We also describe the set of programs $\{|P_i^{\mathrm{H}}\rangle,|P_{i,j}^{\mathrm{H}}\rangle,|P_{i,j}^{\mathrm{C}}\rangle\}$ which apply the single- and multiple-qubit H and CNOT operations on arbitrary qubits on the memory tape, where i and $j\in\mathbb{Z}$:

1. $|P_i^{\mathrm{H}}\rangle$: Apply H to data qubit $D(i)$,

$$|P_i^{\mathrm{H}}\rangle \stackrel{\mathrm{def}}{=} |D_{5i-1}\rangle|\mathrm{H}\rangle. \tag{5}$$

2. $|P_{i,j}^{\mathrm{H}}\rangle$: Apply H to data qubits $D(i:j)$, where $i\geq j$,

$$|P_{i,j}^{\mathrm{H}}\rangle \stackrel{\mathrm{def}}{=} \Pi_{k=j}^{i}|P_k^{\mathrm{H}}\rangle. \tag{6}$$

One could implement this program using a loop but that would require first implementing binary addition of M qubits. Binary addition is possible because one can implement a binary adder such as a Carry Lookahead Adder (CLA) [7] using the NAND program that we defined in [5]. However, since we are only interested in a polynomial order (in the number of qubits) multiple qubit Hadamard transformation program, we define $|P_{i,j}^{\mathrm{H}}\rangle$ as a sequential "unrolled" loop program.

3. $|P_{i,j}^{\mathrm{C}}\rangle$: Apply CNOT to data qubits $D(i)$ and $D(j)$ with $D(i)$ as the control qubit,

$$|P_{i,j}^{\mathrm{C}}\rangle \stackrel{\mathrm{def}}{=} |S_{i,s}\rangle|D_{5j-1}\rangle|\mathrm{CNOT}\rangle|S_{i,s}\rangle. \tag{7}$$

Using the primitive programs defined above, we define $|P_i^{\mathrm{X}}\rangle$ as the program that applies the σ_x (X) operation on data qubit $i\in\mathbb{Z}^+$. Noting that a CNOT operation with the control qubit in the $|1\rangle$ state is equivalent to the X operation, we deduce the equivalence

$$|P_i^{\mathrm{X}}\rangle \equiv |P_{|1\rangle,i}^{\mathrm{C}}\rangle, \tag{8}$$

where the subscript $|1\rangle$ denotes that some suitable data qubit on the memory tape has been prepared in the state $|1\rangle$. Similarly, we define $|P_i^{\mathrm{Z}}\rangle$ as the program that applies the σ_z (Z) operation on data qubit $i\in\mathbb{Z}^+$. Noting that HXH = Z, we deduce

$$|P_i^{\mathrm{Z}}\rangle \equiv |P_i^{\mathrm{H}}\rangle|P_i^{\mathrm{X}}\rangle|P_i^{\mathrm{H}}\rangle. \tag{9}$$

Finally, we define a program $|P_{i,j}^{\mathrm{CZ}}\rangle$ that conditionally applies the Z operation on data qubit $i\in\mathbb{Z}^+$ and data qubit $j\in\mathbb{Z}^+$. Since CNOT is the conditional X operation, we have

$$|P_{i,j}^{\mathrm{CZ}}\rangle \equiv |P_j^{\mathrm{H}}\rangle|P_{i,j}^{\mathrm{C}}\rangle|P_j^{\mathrm{H}}\rangle. \tag{10}$$

UQC Algorithms Using Networked Quantum Oracle Devices

With the notable exception of Shor's factorization algorithm [8], several well known quantum algorithms that achieve a speed-up over their fastest known classical counterparts rely on the use of an oracle, the best known examples being the Deutsch, Deutsch-Jozsa, and Grover algorithms (see Nielsen and Chuang [6], for example). The Deutsch algorithm can determine a global property of a function $f(x)$, namely $f(0)\oplus f(1)$, using only one evaluation of $f(x)$ whereas the fastest classical algorithm requires at least two evaluations of $f(x)$. The Deutsch-Jozsa algorithm can determine whether a two-valued (0 or 1) function $f(x)$ is constant or balanced with only one evaluation of $f(x)$ whereas the fastest classical algorithm requires $2^{n-1}+1$ evaluations, where n denotes the number of bits required to encode the possible values of $f(x)$. Grover's algorithm [9] can find a marked item in an unstructured database of N elements in $O(\sqrt{N})$ operations whereas the fastest classical algorithm requires $O(N)$ operations. Thus, these quantum algorithms all achieve at least a quadratic speedup over their classical counterparts.

These algorithms are well suited to illustrate the use of networked quantum resources with the UQC because they rely on black-box quantum devices that generate some output based on the given input. They thus serve as prototypical examples of a networked quantum node, whose internal implementation details are unknown; only the interface protocol need be known. Here, we assume the simplest protocol, which is that the output is valid one "clock cycle" after making a request.

Deutsch and Deutsch-Jozsa Algorithms on UQC. We now illustrate the use of a networked quantum device in a UQC program by first implementing the simplest known oracle based quantum algorithm, Deutsch's algorithm. The Deutsch oracle works as follows:

$$|x,y\rangle\to\begin{cases}|x,y\oplus f(x)\rangle & \textit{if } |\mathrm{en}\rangle = |1\rangle\,,\\ |x,y\rangle & \textit{otherwise},\end{cases}$$

where f is some function and $|\mathrm{en}\rangle$ denotes the oracle query enable flag. The memory tape is prepared with $D(0)=|0\rangle$ and $D(1)=|1\rangle$ where $D(0)$ and $D(1)$ take the roles of x and y, respectively. We assume without loss of generality that $D(2)$ takes the role of $|\mathrm{en}\rangle$ and is prepared as $|0\rangle$, and $D(3)$ is initially prepared as $|1\rangle$.

The program that executes the Deutsch algorithm is

$$|P_D\rangle \stackrel{\mathrm{def}}{=} |P_{1,0}^{\mathrm{H}}\rangle|S_{2,3}\rangle|S_{2,3}\rangle|P_0^{\mathrm{H}}\rangle|P_h\rangle, \tag{11}$$

where $|P_{1,0}^{\mathrm{H}}\rangle$ applies the Hadamard transform to the data qubits corresponding to x and y. $|S_{2,3}\rangle|S_{2,3}\rangle$ swap qubits $D(2)$ and $D(3)$ thereby setting the oracle's $|\mathrm{en}\rangle$ qubit (recall that $D(2)$ is connected to $|\mathrm{en}\rangle$ and that $D(2)=|0\rangle$ and $D(3)=|1\rangle$ initially) for a single UQC cycle and then clears it, returning the state of $D(3:2)$ back to the original state. At this point, the oracle has generated the output state $|D(0),D(0)\oplus D(1)\rangle$. $|P_0^{\mathrm{H}}\rangle$ then applies the Hadamard transform to the x output of the oracle and $|P_h\rangle$ halts the program thus yielding the following on the memory tape:

$$|D(0),D(1)\rangle = \pm|f(0)\oplus f(1)\rangle\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right].$$

Measuring $D(0)$ yields the result that we were interested in, $f(0)\oplus f(1)$. This is a specific mapping of the gate array implementation of the algorithm (see [6] Figure 1.19, for example) onto the instruction set of UQC.

We can similarly implement the Deutsch-Jozsa algorithm by mapping a gate array implementation such as the one shown in [6], Figure 1.20. In this case, data qubits $D(0:n-1)$ take the role of x, $D(n)$ takes the role of y, and we use $D(n+1)$ as the $|\mathrm{en}\rangle$ qubit. As before, $D(0:n-1)$ are prepared in the $|0\rangle$ state, $D(n)$ is prepared in the $|1\rangle$ state, $D(n+1)$ is prepared in the $|0\rangle$ state and $D(n+2)$ is prepared in the $|1\rangle$ state. The Deutsch-Jozsa oracle works like the Deutsch oracle with the only difference being that x is n qubits wide. The resulting UQC program that computes the Deutsch-Jozsa algorithm is therefore

$$|P_{DJ}\rangle \stackrel{\text{def}}{=} |P^{\mathrm{H}}_{n-1,0}\rangle|S_{n+1,n+2}\rangle|S_{n+1,n+2}\rangle|P^{\mathrm{H}}_{n-1,0}\rangle|P_h\rangle, \quad (12)$$

which is again a direct mapping of the gate array implementation onto the UQC instruction set.

Grover's Algorithm on UQC. We now use the techniques developed in the previous section to implement the Grover unstructured database search algorithm. We assume that the database has only one marked solution as can be determined by using the quantum counting algorithm (see [6] Chapter 6, for example). We denote the query data qubits as $|q\rangle$ and the query enable flag as $|\mathrm{en}\rangle$. The Grover oracle works as follows:

$$|q\rangle \rightarrow \begin{cases} (-1)^{f(q)}|q\rangle & \text{if } |\mathrm{en}\rangle = |1\rangle, \\ |q\rangle & \text{otherwise} \end{cases}$$

where $f(q)=1$ if q is a solution to the search problem and $f(q)=0$ otherwise. More concisely, the oracle performs the unitary transformation

$$U_m \stackrel{\text{def}}{=} \mathbb{I}-2|m\rangle\langle m|, \quad (13)$$

where $|m\rangle$ denotes the marked solution. In other words, the oracle flips the phase of the solution state but leaves non-solution states unchanged. Grover's algorithm prepares an initial query state as the equal superposition of all elements in the database, followed by $O(\sqrt{2^n})$ iterations of G, where

$$G \stackrel{\text{def}}{=} (2|s\rangle\langle s|-\mathbb{I})U_m, \quad (14)$$

and

$$|s\rangle = \frac{1}{\sqrt{2^n}}\sum_{i=0}^{2^n-1}|i\rangle \quad (15)$$

denotes the equal superposition of all database elements.

Thus, the first step in the program is to create a superposition of all database items in $D(n:1)$ where $D(i)=M(5i-1)$, $i\in\mathbb{Z}^+$, as the first query input. This is accomplished by the multiple qubit Hadamard primitive program $|P^{\mathrm{H}}_{n,1}\rangle$ defined in Eq. (6). The next step is to perform an oracle query. The following program performs an oracle call with query data prepared in $D(n:1)$:

$$|P_m\rangle \stackrel{\text{def}}{=} |S_{n+1,n+2}\rangle|S_{n+1,n+2}\rangle, \quad (16)$$

where $D(n+1)$ is used as the oracle query enable qubit and $D(n+2)$ is initialized to $|1\rangle$. $D(n+1)$ is assumed to be initialized to $|0\rangle$ (i.e. the oracle query data is disabled at start-up). This program simply sets the query enable qubit for a single UQC cycle and then clears it, returning the state of $D(n+2:n+1)$ back to the original state. Thus, upon running $|P_m\rangle$, the result of the oracle call is in $D(n:1)$, i.e. this program is functionally equivalent to U_m.

The next step is to implement a program $|P_s\rangle$ that performs the reflection of a given state about the superposition of all basis states $|s\rangle$. This requires a conditional-phase operation that works as follows:

$$|x\rangle \rightarrow \begin{cases} |x\rangle & \text{if } x=0, \\ -|x\rangle & \text{otherwise} \end{cases}$$

where $|x\rangle$ is n qubits wide. Up to a global phase, this can be implemented using the following procedure:

1. Apply the σ_x operation to all n qubits.
2. Apply a controlled-$\mathbf{Z}$ operation using $n-1$ qubits as control qubits and the remaining qubit as the data qubit.
3. Apply the σ_x operation to all n qubits.

We can construct a multiple qubit controlled-$\mathbf{Z}$ program $|P^{\mathrm{CZ}}_{i,j,k}\rangle$ where qubits i through j are the control qubits and qubit k is the data qubit, with the $|P^{\mathrm{CZ}}_{i,j}\rangle$ program defined in Eq. (10) and the Toffoli program $|P^{\mathrm{Toff}}_{i,j,k}\rangle$ that we defined in [5] using a procedure analogous to that described in [6], Chapter 4. Armed with $|P^{\mathrm{CZ}}_{i,j,k}\rangle$, we construct $|P_s\rangle$ as follows:

$$|P_s\rangle \stackrel{\text{def}}{=} |P^{\mathrm{H}}_{n,1}\rangle|P^{\mathrm{X}}_{n,1}\rangle|P^{\mathrm{CZ}}_{2,n,1}\rangle|P^{\mathrm{X}}_{n,1}\rangle|P^{\mathrm{H}}_{n,1}\rangle. \quad (17)$$

It can be readily verified that this is functionally equivalent to the $2|s\rangle\langle s|-\mathbb{I}$ operator. Thus, a program that performs a single Grover iteration is

$$|P_G\rangle \stackrel{\text{def}}{=} |P_m\rangle|P_s\rangle. \quad (18)$$

In summary, the complete program to search a database of 2^n items with a single marked solution is

$$|G\rangle \stackrel{\text{def}}{=} |P^{\mathrm{H}}_{n,1}\rangle(|P_G\rangle)^{N_G}|P_h\rangle, \quad (19)$$

where $N_G=\frac{\pi}{4}\sqrt{2^n}$ is the number of Grover iterations that can be pre-computed based on the database size, or that UQC can compute from the database size using a classical algorithm. Upon execution of $|G\rangle$, a measurement of $D(n:1)$ reveals the solution $|m\rangle$. Because there are no oracle queries associated with $|P^{\mathrm{H}}_{n,1}\rangle$ and $|P_h\rangle$, we immediately identify the complexity (as a measure of the number of oracle queries) of $|G\rangle$ as N_G. As is to be expected, this complexity is identical to the number of oracle queries associated with an implementation in the gate array framework.

Discussion

We have presented a scheme to allow universal quantum computers to utilize networked quantum resources. We have illustrated the scheme by devising *UQC* programs that implement the well-known oracle based Deutsch, Deutsch-Jozsa, and Grover algorithms using networked quantum oracle devices. We have therefore demonstrated that universal quantum computers can access networked quantum devices in a way analogous to that by which classical computers access network resources. The method that we used to map quantum algorithms onto *UQC* can be applied to implement and analyze other quantum algorithms.

Author Contributions

Wrote the paper: AL. Assisted in the development and analysis of the universal quantum computer: ML LvS.

References

1. Kimble HJ (2008) The quantum internet. Nature 453: 1023–1030.
2. Curcic T, Filipkowski ME, Chtchelkanova A, D'Ambrosio PA, Wolf SA, et al. (2004) Quantum networks: from quantum cryptography to quantum architecture. SIGCOMM Comput Commun Rev 34: 3–8.
3. Griffiths DJ (2005) Introduction To Quantum Mechanics. Pearson Education Inc., second edition.
4. Deutsch D (1985) The church-turing principle and the universal quantum computer. Proc R Soc London 400: 97–117.
5. Lagana AA, Lohe MA, von Smekal L (2009) Construction of a universal quantum computer. Phys Rev A 79: 052322.
6. Nielsen MA, Chuang IL (2000) Quantum Computation and Quantum Information. Cambridge University Press.
7. Waser S, Flynn MJ (1982) Introduction To Arithmetic for Digital Systems Designers. Holt, Reinhart and Winston.
8. Goldwasser S, ed (1994) Proceedings of the 35th Annual Symposium on the Foundations of Computer Science. IEEE Computer Society.
9. Grover LK (1997) Quantum mechanics helps in searching for a needle in a haystack. Phys Rev Lett 79: 325–328.

References

[1] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000. 1, 10, 21, 36, 52, 53, 54, 55, 57, 67, 72

[2] S. Goldwasser, editor. *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*. IEEE Computer Society, 1994. 1, 53

[3] T.D. Kieu. **Quantum Algorithms for Hilbert's Tenth Problem**. *Int. J. Theor. Phys.*, **42**:1451–1468, 2003. 1, 5

[4] N. Bhattacharya, H. B. van Linden van den Heuvell, and R. J. Spreeuw. **Implementation of Quantum Search Algorithm using Classical Fourier Optics**. *Physical Review Letters*, **88**(13):137901–, 2002. 1

[5] Lov K. Grover. **Quantum Mechanics Helps in Searching for a Needle in a Haystack**. *Phys. Rev. Lett.*, **79**(2):325–328, Jul 1997. 1, 53, 59

[6] Samuel L. Braunstein and Arun K. Pati. **Speed-up and entanglement in quantum searching**. *Quantum Info. Comput.*, **2**:399–409, August 2002. 1

[7] David A. Meyer. **Sophisticated Quantum Search Without Entanglement**. *Phys. Rev. Lett.*, **85**:2014–2017, Aug 2000. 1

[8] Noah Linden and Sandu Popescu. **Good Dynamics versus Bad Kinematics: Is Entanglement Needed for Quantum Computation?** *Phys. Rev. Lett.*, **87**(4), 2001. 1

[9] D. Deutsch. **Quantum computational networks**. *Proc. R. Soc. London*, **425**, 1989. 1

[10] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. **Quantum Computation by Adiabatic Evolution**. *e-print at* `arXiv:quant-ph/0001106v1`, 2000. 2

[11] Ari Mizel, M. W. Mitchell, and Marvin L. Cohen. **Energy barrier to decoherence**. *Phys. Rev. A*, **63**:040302, Mar 2001. 2

[12] Paolo Zanardi and Mario Rasetti. **Holonomic quantum computation**. *Physics Letters A*, **264**(2-3):94 – 99, 1999. 2

[13] Michael A. Nielsen. **Quantum computation by measurement and quantum memory**. *Physics Letters A*, **308**(2-3):96 – 100, 2003. 2

[14] Edward Farhi and Sam Gutmann. **Quantum computation and decision trees**. *Phys. Rev. A*, **58**:915–928, Aug 1998. 2

[15] D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev. **Adiabatic Quantum Computation is Equivalent to Standard Quantum Computation**. `arXiv:0405098v2`, 2005. 2

[16] Ari Mizel, Daniel A. Lidar, and Morgan Mitchell. **Simple Proof of Equivalence between Adiabatic Quantum Computation and the Circuit Model**. *Phys. Rev. Lett.*, **99**:070502, Aug 2007. 2

[17] R. K. Saurya Das and G. Kunstatter. **Energy and efficiency of adiabatic quantum search algorithms**. *Journal of Physics A Mathematical General*, **36**:2839–2845, 2003. 2, 3

[18] M V Berry. **Quantal Phase Factors Accompanying Adiabatic Changes**. *Proceedings of the Royal Society A Mathematical Physical and Engineering Sciences*, **392**(1802):45–57, 1984. 3

[19] Robert Raussendorf and Hans J. Briegel. **A One-Way Quantum Computer**. *Phys. Rev. Lett.*, **86**:5188–5191, May 2001. 4

[20] Michael A. and Nielsen. **Cluster-state quantum computation**. *Reports on Mathematical Physics*, **57**(1):147 – 161, 2006. 4

[21] D. Deutsch. **The Church-Turing Principle and the Universal Quantum Computer**. *Proc. R. Soc. London*, **400**(1818):97–117, July 1985. 4, 15, 17, 21, 50

[22] John M. Myers. **Can a Universal Quantum Computer Be Fully Quantum**. *Phys. Rev. Lett.*, **78**(78):1823–1824, March 1997. 5, 15, 18

[23] A. A. Lagana, M. A. Lohe, and L. von Smekal. **Construction Of A Universal Quantum Computer**. *Phys. Rev. A*, **79**(5):052322, May 2009. 5, 42, 67

[24] Mike Stannett. **X-Machines and the Halting Problem: Building a Super-Turing Machine.** *Formal Asp. Comput.*, **2**(4):331–341, 1990. 5

[25] Martin Ziegler. **Computational Power of Infinite Quantum Parallelism**. In *pp.2057–2071 in International Journal of Theoretical Physics vol.44:11*, 2005. 5

[26] H. J. Kimble. **The quantum internet**. *Nature*, **453**(7198):1023–1030, 06 2008. 5, 49

[27] Tatjana Curcic, Mark E. Filipkowski, Almadena Chtchelkanova, Philip A. D'Ambrosio, Stuart A. Wolf, Michael Foster, and Douglas Cochran. **Quantum networks: from quantum cryptography to quantum architecture**. *SIGCOMM Comput. Commun. Rev.*, **34**(5):3–8, 2004. 5, 49

[28] Antonio A. Lagana, Max A. Lohe, and Lorenz von Smekal. **Interfacing External Quantum Devices to a Universal Quantum Computer**. *PLoS ONE*, **6**(12):e29417, 12 2011. 6

[29] Christof Zalka. **Grover's quantum searching algorithm is optimal**. *Phys. Rev. A*, **60**(4):2746–2751, Oct 1999. 6, 59

[30] Stephen D. Bartlett, Hubert de Guise, and Barry C. Sanders. **Quantum encodings in spin systems and harmonic oscillators**. *Physical Review A*, **65**(52316), 2001. 9, 12

[31] Toby Ord. **Hypercomputation: computing more than the Turing machine**. `http://arxiv.org/pdf/math.LO/0209332`, 2002. 10

[32] Yu Shi. **Remarks on universal quantum computer**. *Phys. Lett. A*, **293**:277–282, 2002. 15, 17, 19

[33] Masanao Ozawa. **Quantum Nondemolition Monitoring of Universal Quantum Computers**. *Phys. Rev. Lett.*, **80**(3):631–634, January 1998. 15, 19, 40

[34] N. S. Yanofski and M. A. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, 2008. 20

[35] E. Bernstein and U. Vazirani. **Quantum complexity theory**. *SIAM Journal on Computing*, **26**(5):1411–1473, 1997. 20

[36] Satoshi Iriyama, Takayuki Miyadera, and Masanori Ohya. **Note on a universal quantum Turing machine**. *Phys. Lett. A*, **372**:5120–5122, 2008. 20

[37] Shlomo Waser and Michael J. Flynn. *Introduction To Arithmetic for Digital Systems Designers*. Holt, Reinhart and Winston, 1982. 35

[38] A.M. Turing. **On computable numbers, with an application to the Entscheidungsproblem**. **42** of *2*, pages 230–265. Proc. London Math. Soc., 1936. 40

[39] A. Church. **n unsolvable problem of elementary number theory**. *Amer. J. Math.*, **58**:345–363, 1936. 40

[40] M. A. Nielsen. **Computable Functions, Quantum Measurements, and Quantum Dynamics**. *Physical Review Letters*, **79**(15):2915–2918, October 1997. 41, 42

[41] Takayuki Miyadera and Masanori Ohya. **On Halting Process of Quantum Turing Machine**. *Open Systems and Information Dynamics*, **12**:261–264, 2005. 10.1007/s11080-005-0923-2. 41

[42] Michel le Bellac. *Quantum Physics*. Cambridge University Press, 2006. 43

[43] David J. Griffiths. *Introduction To Quantum Mechanics*. Pearson Education Inc., second edition, 2005. 49

[44] V. E. Korepin and Y. Xu. **Binary quantum search**. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, **6573** of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, May 2007. 59

[45] Vladimir E. Korepin and Brenno C. Vallilo. **Group Theoretical Formulation of Quantum Partial Search Algorithm**. *Progress of Theoretical Physics*, **116**:783, 2006. 59

[46] Vladimir E Korepin. **Optimization of partial search**. *Journal of Physics A: Mathematical and General*, **38**(44):L731–L738, 2005. 59

[47] Byung-Soo Choi and Vladimir E. Korepin. **Quantum Partial Search of a Database with Several Target Items**. *Quantum Information Processing*, **6**(4):243–254, 2007. 59

[48] Vladimir E. Korepin and Jinfeng Liao. **Quest for Fast Partial Search Algorithm**. *Quantum Information Processing*, **5**(3):209–226, 2006. 59

[49] Vladimir E. Korepin and Lov K. Grover. **Simple Algorithm for Partial Quantum Search**. *Quantum Information Processing*, **5**(1):5–10, 2006. 59

[50] Byung-Soo Choi, Thomas A. Walker, and Samuel L. Braunstein. **Sure Success Partial Search**. *Quantum Information Processing*, **6**(1):1–8, 2007. 59, 68

[51] J. Radhakrishnan L.K. Grover. **Is partial quantum search of a database any easier?** In *SPAA '05: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages pp. 186–194, New York, NY, USA, 2005. ACM. 59, 60

[52] Daniel S. Abrams and Seth Lloyd. **Nonlinear Quantum Mechanics Implies Polynomial-Time Solution for NP-Complete and $\sharp$ P Problems**. *Phys. Rev. Lett.*, **81**(18):3992–3995, Nov 1998. 69