Low-depth quantum architectures

Paul Pham

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2013

Reading Committee:

Aram Harrow, Chair

Paul Beame

Mark Oskin

Boris Blinov

Program Authorized to Offer Degree:
University of Washington Computer Science & Engineering

University of Washington

Abstract

Low-depth quantum architectures

Paul Pham

Chair of the Supervisory Committee:
Affiliate Professor Aram Harrow
Computer Science & Engineering

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

QUANTUM ARCHITECTURE: the study of how to lay out quantum bits and their allowed interactions in a physical system to execute quantum algorithms efficiently in time, space, and other resources.

ACKNOWLEDGMENTS

DEDICATION

To family, teachers, friends, and muses.

This is for everyone who believed that I could do this.

Chapter 0
INTRODUCTION

The combination of existing scientific disciplines has often been the cradle and the playground for entirely new discoveries. Quantum computing is one such unification of two seemingly disparate fields. Computer science and quantum physics originated as contemporaries in the early part of the 20th century, both culminating in technology that decisively ended World War II. They have remained largely independent until recently, but the common thread connecting them has always existed: engineering.

From the perspective of computer science, engineering is problem-driven. We humans wish to solve a problem, like calculating ballistic weapon trajectories, and we work backwards to electromechanical relays or punched cards. To calculate, we need to perform fast arithmetic on stored numbers, for which we need to design devices in a certain configuration, for which we need certain materials from nature. At heart, computer science is the study of structured problem-solving which historically has had a strong mechanical bias. This perspective asks the questions: what do we want to do, what is desirable, and how can we build it? It is like a child going into a LEGO[1] store with a pre-existing plan: the child only buys the bricks needed, or orders custom bricks. It is an efficient plan for brick procurement, but not necessarily creative in exploring the space of all possible designs.

From the perspective of quantum physics, engineering is phenomena-based. We humans wish to describe reality and observe nature. We discover objects and effects in nature, and only afterwards may we then discover that they can perform some useful task. An example is the discovery of quantum tunneling and its subsequent use to build a field-effect transistor (FET) for digital electronics. Physics asks the questions: what is available to us, what is possible, and what could we do it? It is like a child going into a LEGO store with no pre-conceived plan, investigating all the parts, dreaming up ways to

[1] Trademarked.

use them, or even delving into custom brick design at a more fundamental level. This will generate completely new designs and models, but it is much slower than brick-shopping.

These perspectives are intertwined and serve complementary purposes, and we often alternate between the two of them. In our transistor example, we probably would not have thought to build a FET if we had not already needed fast electrical switching, for which vacuum tubes were proving inadequate. Moreover, we can go beyond using quantum physics to simulate a classical device. There is a remarkable new class of algorithms designed using the building blocks of quantum physics. The first instance of such an algorithm solves the problem of factoring.

The problem of factorization, or factoring, is simple to state and difficult to solve. Given an integer m, return the prime integers $\{p_1, \ldots, p_k\}$ whose product of their powers is m: $m = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$. It is a beautiful theoretical problem posed by Carl Friedrich Gauss in 1801; although he is considered by many to be one of history's greatest mathematicians, he was still unable to factor arbitrarily large numbers. His feelings on the subject can be paraphrased as follows: "Factoring was the most important problem in the theory of numbers, itself the crown jewel of mathematics, itself the queen of the sciences."[2] However, it is a quirk of cryptography that the hardness of problems can be used as an advantage. Therefore, factoring also has immense practical usefulness as the basis of the RSA cryptosystem, the most widely-used cryptosystem around the world and on the internet.

The somewhat sinister political importance of factoring to governments is the ability to gather intelligence on each other and on their citizens. The importance to citizens is the ability to choose security parameters so that their correspondence is protected from each other and their governments for a limited time. The organizers of the popular democratic uprisings in the Middle East known as the Arab Spring in 2011 definitely benefited from using cryptosystems. RSA was secure against compromise given the resources of the regimes that were toppled. A government able to hack the organizers' Twitter accounts or Facebook pages could misdirect protest actions or identify activists for imprisonment.

---

[2] *Gauss zum Gedächtniss* (1856) by Wolfgang Sartorius von Waltershausen

Similarly, being able to accurately estimate the resources needed to factor an RSA key of a certain size may help protect future citizens against governments with a quantum computers, as governments will be the first organizations interested in and able to afford such a device.

How can we reason about the computational resources needed to solve the problem of factoring an n-bit number? Before 1994, the best known algorithm was the number field sieve which has a running time of $exp(\sqrt[3]{c \log n (\log \log n)^2})$ [61]. This can be considered a pre-quantum algorithm in a pre-quantum era, when computer scientists could remain unaware of their future with quantum physics. The number field sieve's sub-exponential (but still super-polynomial) running time is intractable. This conjectured hardness allows users and attackers of the RSA cryptosystem to experience the kind of dramatic tension above while at the same time giving hope to mathematicians of some future improvement.

In 1994, Peter Shor fulfilled this hope in a landmark discovery using the more general computational power of quantum physics to factor in polynomial time: $O(n^3 \log n \log \log n)$ [87]. This kickstarted the entire field of quantum computing; united many disciplines including computer science, physics, engineering, and mathematics; and provided the foundation for this dissertation. The computational model in which this remarkable feat was achieved made some assumptions about quantum computing devices, which are now standard within the field but which engender skepticism outside it. These assumptions include robust error-correction (to preserve fragile quantum information against environmental noise) and high-fidelity quantum gates (to exert reliable classical control). Whether these assumptions can be realized is an active area of experimental physics research, but this is not the topic of this current work. However, given that these assumptions will eventually be realized, how can we optimize Shor's factoring algorithm? To answer this question, we need to characterize quantum circuits.

Analogous to the classical circuit model, a quantum circuit consists of a network of gates operating on quantum bits (qubits). A quantum circuit possesses certain quantifiable properties known as *resources*, such the time it takes to complete (circuit depth), how many gates it contains (circuit size), and how many quantum bits it operates on (circuit width). See Figure 1 for the standard graphical representation of a quantum circuit with

htb!

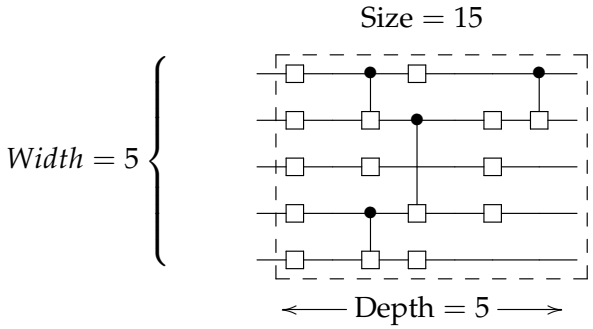

Figure 1: A quantum circuit with single-qubit and two-qubit gates and with resources of depth, size, and width.

the resources marked, which will often refer to simply as depth, size, and width. We will define these resources more formally later. The horizontal lines represent qubits, which begin in some input state (usually initialized to all $|0\rangle$'s) on the left and resulting in some output state on the right.

Because quantum computation is a rapidly maturing field, we cannot provide an adequate background to quantum circuits from first principles for novices. The interested beginner is referred to the standard textbook in the field by Nielsen and Chuang [76] or the course notes by Dave Bacon,[3] especially for a primer on quantum circuits. Building upon this background, we can provide an introduction into the subfield of quantum architecture with the current work.

Quantum architecture is both a thing and a study of that thing. A quantum architecture (as a thing) is a quantum circuit with constraints. For this dissertation, we will concern ourselves with two main constraints: (1) the layout graph of qubits which constrain a fixed two-qubit gate interactions and (2) the decomposition of all single-qubit gates to a discrete, finite, set of gates which are universal when combined with a fixed two-qubit gate. Quantum architecture provides an intermediate layer in between working with actual hardware (the physical technology in a laboratory) and abstract theory

[3] http://www.cs.washington.edu/education/courses/cse599d/06wi/

(algorithms divorced of all physical constraints). As quantum architects, we can optimize algorithms in a general way which is relevant to many kinds of laboratory experiments. At the same time, we can take into account the more complicated nature of physical devices to provide more pessimistic, accurate, numerical resource estimates than more high-level, theoretical results.

The role of quantum architecture (as a study) is two-fold. First, to quantify and improve the circuit resources required to run a quantum algorithm on a system of varying degrees of realism. Second, to provide tradeoffs between circuit resources, or conceptual knobs used to tune a particular implementation. This allows experimentalists to configure a quantum architecture (the thing) to meet their needs in a laboratory while allowing theorists to improve tradeoffs or design new algorithms which take physical constraints into account.

We are now ready to state the thesis of this dissertation:

> Studying the depth of quantum architectures will help us design quantum computers to solve human problems within a human lifetime.

In particular "a human lifetime" means on the order of 50 years, which is the expected expected lifetime for the author. This not only means that an implementation of a quantum architecture in a laboratory should complete within 50 years, but that all resources needed to build and power such a laboratory should also be acquirable in 50 years.

"Human problems" in this dissertation is exemplified by the special-case of factoring. We will optimize the circuit resources for Shor's algorithm mapped to a specific quantum architecture with the hope of discovering general principles that can be applied to other quantum algorithms (and other human problems) in the future.

"Studying the depth" of quantum architectures is meant to make progress towards answering several questions. First, is it possible to improve circuit depth, at the cost of all other circuit resources, for a factoring architecture beyond currently known results? Second, what is the optimal depth possible? Third, what tradeoffs are introduced between depth and other circuit resources? Finally, what is the most relevant of these tradeoffs?

Achieving minimal depth may require materials or energy beyond human abilities to acquire within the 50 years quoted above. Even ignoring the important concerns of scalable error-correction and high-fidelity control, we can already answer the question "Can a quantum factoring machine be built?" by upper-bounding physical resources for the best known architectures, the ones presented in this work.

Therefore, in support of this thesis, our dissertation will investigate both low-depth factoring architectures as well as characterizing their circuit resource tradeoffs. Our results will represent progress towards a more realistic, configurable quantum factoring architecture.

The remainder of this chapter is devoted to the necessary background needed to understand quantum architecture in general and factoring architectures in particular. Quantum architecture has potentially many concerns, but we focus on two of them: interaction distance and fault-tolerant quantum compiling. Most importantly, we do not examine the resources required for error-correction and we assume that all qubits are logical qubits and the operations on them are logical operations. However, our results will also directly benefit error-correcting architectures. We also do not model properties that are extremely specific to physical technologies, although eventually a fault-tolerant quantum architecture will need to commit to such a technology. Readers interested in integrated, fault-tolerant, technology-specific papers are referred to Jones et al. for quantum dots [50] and Kreger-Stickles and Oskin for ion traps [58].

Without loss of generality, the gates of a quantum circuit can be reduced to single-qubit and two-qubit gates, called a circuit basis. In fact, we can use a fixed two-qubit gate and combine it with arbitrary single-qubit gates. Section 0.1 provides the necessary background for understanding circuit bases and single-qubit gates, which do not impose any constraints on which qubits can interact with each other. Section 0.2 defines quantum architectures and architectural models in more detail using a fixed, finite basis and imposing connectivity constraints on two-qubit gates (also described as *interactions*). Section 0.3 introduces recent constant-depth techniques for simulating long-distance interactions with nearest-neighbor interactions, using linear size and width. Constant-depth communication introduces a tradeoff between depth and the other two resources. Specifically,

reducing depth raises the possibility that more circuit resources are devoted to communication than to computation. This tradeoff is examined in more detail in Section 0.4 by introducing the notion of a module and hybrid nearest-neighbor quantum architectures. Finally, Section 0.5 gives an overview of the remaining chapters in this dissertation, placing them within the context of this introduction and our thesis statement above.

0.1 Quantum Gates and Circuit Bases

To compile, or implement, arbitrary quantum algorithms, we must construct circuits out of gates from a universal set, which we call a *circuit basis*, or just *basis*. This should not be confused with a basis for a vector space. Therefore, we will now review quantum gates and how to combine them into circuits. This procedure is known as quantum compiling, but we will not delve into its details until we need them in Chapter 3.

A quantum gate on n-qubits is a $2^n \times 2^n$ unitary matrix (an element of $U(2^n)$). We can consider this the overall circuit width. Often, we find it useful to neglect a global phase, since these cannot be measured in quantum mechanics. However, a global phase on a particular system S may result in a measurable relative phase in a large system S' of which S is a subsystem. Therefore, for our purposes we will only distinguish between $U(2^n)$ and $SU(2^n)$ in the few cases where it matters for quantum compiling. The distinction between a quantum circuit and a quantum gate is relative; often we consider a quantum gate as a fundamental primitive of our physical technology, and a circuit as a composite of these gates corresponding to a quantum algorithm.

In Section 0.1.1 we will review the Pauli single-qubit gates and their corresponding group. In Section 0.1.2 we will introduce the Clifford group. In Section 0.1.3 we will introduce controlled operations and the Toffoli gate. In Section 0.1.4 we will discuss *single-qubit compiling* and how a general single-qubit gate can be compiled into rotations about Bloch sphere axes. In Section 0.1.5 we will present distance metrics to measure the quality of our single-qubit (and later multi-qubit) approximations. In Section 0.1.6 we will finally define what it means for a gate set to be universal.

0.1.1 Pauli Group

We review here the Pauli group on one qubit, $\mathcal{P}_1 = \{I, X, Y, Z\}$. These last three represent rotations of π on the Bloch sphere about the x-axis, y-axis, and z-axis, using the homomorphism between $SU(2)$ and $SO(3)$. The 2×2 identity matrix is denoted I. The group $\mathcal{P}_1$ also serves as a complex vector basis for generating elements of $U(2)$.

$$X{=}\sigma_x{=}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad Y{=}\sigma_y{=}\begin{bmatrix} 0 & i \\ -i & 0 \end{bmatrix} \tag{1}$$

$$Z{=}\sigma_z{=}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad I{=}\sigma_0{=}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2}$$

We define the Pauli group $\mathcal{P}_n$ on n qubits as the set of all n-qubit operators which are tensor products of elements from $\mathcal{P}_1$.

0.1.2 The Clifford Group

We define the normalizer of $\mathcal{P}_n$ as the Clifford group $\mathcal{C}_n$ on n qubits.

$$\mathcal{C}_n = \{C \in U(2^n) | CPC^\dagger \in \mathcal{P}_n \quad \forall P \in \mathcal{P}_n\} \tag{3}$$

Of particular interest to us is the two-qubit Clifford group $\mathcal{C}_2$, which is generated by the following matrices:

$$\mathcal{C}_2 = \langle H, S, CNOT \rangle \tag{4}$$

The first two Clifford generator matrices are single-qubit gates (2×2 unitary matrices) and their inclusion means they can be applied on either the first or the second qubit.[4] The matrix H is known as the Hadamard gate, and it is a special case of the general Walsh-Hadamard transform. It is its own adjoint: $H^\dagger = H$. The matrix S is known as the phase gate, and it can be considered the "square root" of the Pauli Z gate (up to a phase):

[4]Standard convention writes H_i to mean H on qubit i and likewise for S_i.

$S^2 = Z$. Equivalently, it can be viewed as a $\pi/2$ rotation about the Bloch sphere z-axis, and its adjoint $S^\dagger$ is the reverse rotation of $-\pi/2$. These matrices are defined below.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \qquad S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \tag{5}$$

The Hadamard matrix also has the special property that it changes between the X basis and the Z basis, that is, the vector basis for single-qubit states consisting of eigenstates of the Pauli X and Pauli Z gates, respectively. In fact, using the identities $X = HZH$ and $S^2 = Z$, it is easy to see why X and Z are often listed as generators of the Clifford group as well.

The last Clifford generator matrix is a two-qubit gate (a 4×4 unitary matrix) which also represents a *controlled* operation. That is, based on the $|1\rangle$ component of the *control* qubit, it applies a single-qubit gate (in this case, Pauli X) to the *target* qubit. In fact, both $CNOT$ and X are also fundamental gates in classical reversible logic as well, where X is also the Boolean NOT gate on classical bits. That is why the gate is called $CNOT$, for "controlled-NOT." Its inclusion in the generating set for $\mathcal{C}_2$ means that it can be applied in either direction: with control on qubit 1 and target on qubit 2 or vice versa. $CNOT$ is defined below.

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{6}$$

Likewise, the general Clifford group on n qubits $\mathcal{C}_n$ can be generated from the same set as $\mathcal{C}_2$, with gates understood to apply to any of the n qubits. With three $CNOT$'s, we can implement the $SWAP$ gate which exchanges the states of two qubits.

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

The gate $CNOT$ has historical importance in quantum computing partly due to its use in many early quantum gate decompositions and its ability to be performed fault-tolerantly in many physical technologies. It will be our primary two-qubit gate. Along with arbitrary single-qubit gates, it is universal for quantum computation [5]. Therefore, we will give this basis a special name:

$$\mathcal{Q} = \{U(2) \cup CNOT\} \tag{8}$$

0.1.3 Controlled Gates

The principle of a controlled gate can be generalized to multiple controls using the "meta-operator" notation from [53]. By $\Lambda^n(U)$, we mean an $(n+1)$-qubit gate ($2^{n+1} \times 2^{n+1}$ unitary matrix) with n control qubits and a single-qubit target gate $U \in U(2)$. An important multiply-controlled gate, which is universal for classical reversible circuits, is the Toffoli gate, or controlled-controlled-NOT.

$$Toffoli = \Lambda^2(X) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{9}$$

As seen above, multiply-controlled single-qubit gates $\Lambda^n(U)$ have a special, sparser structure than general n-qubit gates in $U(2^n)$. These play a special role in many multi-qubit decompositions, about which we will say more in Chapter 3.

There is also a special case of a "targetless" controlled single-qubit gate which simply rotates the $|1\rangle$ component of a single-qubit state.

$$\Lambda(e^{i\phi}) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix} \tag{10}$$

This gate is a key tool and simplification for single-qubit compiling.

0.1.4 Single-Qubit Compiling

A seemingly simpler task than general compiling is single-qubit compiling. This will illustrate the basic principles of quantum compiling and the structure that we will exploit later to choose an effective basis. Moreover, it will reveal a general relationship between many of the single-qubit gates that we have already introduced.

First, we review how to decompose a general $U \in U(2)$ into three single-qubit rotations about the Bloch sphere x-axis and z-axis, the so-called Euler angle decompositions [76]. This gives rise to a factor of 3 which commonly appears in resource calculations in the literature.

$$U = e^{i\delta} R_Z(\gamma) R_X(\beta) R_Z(\alpha) \tag{11}$$

The gate $R_Z(\phi)$ represents a rotation about the Bloch sphere z-axis, of which the Pauli Z gate is a special case of a π rotation. In fact, it is the same as the controlled-phase gate we introduced in the previous section, up to a global phase.

$$R_Z(\phi) = \begin{bmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{bmatrix} = e^{-i\phi/2} \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix} = e^{-i\phi/2} \Lambda(e^{i\phi}) \tag{12}$$

We can now state the relationship between S and Z, as well as introduce an important new gate T which is the square root of S up to a phase. All three are rotations about the Bloch z-axis by power-of-two fractions of π.

$$Z = R_Z(\pi) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad S = R_Z(\tfrac{\pi}{2}) = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \qquad T = R_Z(\tfrac{\pi}{4}) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{13}$$

Likewise, the gate $R_X(\phi)$ represents a rotation about the Bloch sphere x-axis, of which the Pauli X gate is a special case of a π rotation.

$$R_X(\phi) = \begin{bmatrix} \cos\frac{\phi}{2} & -i\sin\frac{\phi}{2} \\ -i\sin\frac{\phi}{2} & \cos\frac{\phi}{2} \end{bmatrix} \tag{14}$$

Similar decompositions can be given in terms of R_X and R_Y, or in terms of R_Y and R_Z. Solving for the angles $\{\alpha, \beta, \gamma, \delta\}$ involves writing four equations in four variables, which can be found in the standard textbook [76]. We will not say more about their solution here, except that we can implement the global phase shift $e^{i\delta}$ using the identities below, which are adapted from [53].

$$e^{i\delta} = R_Z(\phi) \cdot X \cdot R_Z(\phi) \cdot X \tag{15}$$

$$X = R_X(\pi) \tag{16}$$

$$Z = R_Z(\pi) \tag{17}$$

$$R_X(\phi) = H \cdot R_Z(\phi) \cdot H \tag{18}$$

It now seems that a reasonable basis for single-qubit compiling are arbitrary $R_Z(\phi)$ and $R_X(\phi)$ rotations, along with H. However, in practice our classical control can only implement rotations with finite precision. How can we measure this, or any other, precision arising from approximation?

0.1.5 Distance Metrics

Each gate from our basis is a unitary matrix of bounded dimension, and the action of an entire n-qubit compiled circuit $\tilde{C}$ is also a $2^n \times 2^n$ unitary matrix. This matrix can be formed by the product of $2^n \times 2^n$ matrices G_i which are a tensor product of gate matrices from our basis operating on individual qubits. Our desired target matrix C is itself a matrix from $U(2^n)$, and therefore we will need a distance metric that operates on matrices (specifically, the difference of matrices).

$$\tilde{C} = G_D G_{D-1} \cdots G_2 G_1 \tag{19}$$

One distance metric used in theoretical literature is the operator norm of a matrix M, is defined as the maximum amount it scales the vector norm of all unit-length vectors.

This is sometimes also called the infinity-norm, or supremum-norm (sup-norm).

$$\|M\|_\infty = \max_{\||v\rangle\|=1} \|M\,|v\rangle\,\| \tag{20}$$

However, this is not an operational definition. Moreover, we often wish to neglect a global phase in a unitary matrix, which is not measurable in quantum physics. To measure phase-independent distance between two unitary matrices, we can use the following distance measure due to Fowler [36].

$$dist(U,V) = \|U - V\| \equiv \sqrt{\frac{2^n - |tr(U^\dagger V)|}{2^n}} \tag{21}$$

Now can quantify the quality of our approximations through an error ϵ.

$$\|C - \tilde{C}\| = \|C - G_D \cdots G_1\| < \epsilon \tag{22}$$

Often ϵ will be small, and we will upper bound it by some power of $\frac{1}{2}$. Therefore, we define a new parameter which is the number of bits needed to encode the exponent of this increasingly small fraction.

$$\epsilon = \frac{1}{2^k} \qquad k = \log(1/\epsilon) \tag{23}$$

It is natural to suppose that compiling better approximations requires more resources, and these resources are expressed as functions of these parameters, increasing with decreasing ϵ and increasing k. In fact, often the efficiency and the capabilities of a quantum compiler depend on its basis. Therefore, we conclude this section by discussing circuit bases.

0.1.6 Circuit Bases

Definition 1. ***Circuit basis.*** *A basis for a quantum circuit (family) is a universal set of bounded-qubit gates (usually operating on no more than 3 qubits each). We call a basis* finite *if it contains a finite number of gates; that is, it contains discrete gates and not an infinite continuum of gates. We call a basis* fixed *if its members are independent of the number of qubits in the input circuit (its width).*

For fault-tolerant quantum computing, we are interested in compiling circuits to a fixed, finite basis. What does it mean for a fixed, finite basis to be universal for an infinite group like $SU(2^n)$?

Definition 2. ***Universal approximation.*** *We call a fixed, finite set of gates $\mathcal{G}$* universal *for a group $U(2^n)$ iff for every desired target $C \in U(2^n)$ and desired error ϵ, we can return a sequence of gates $(g_1, g_2, \ldots, g_S)$ from $\mathcal{G}$ where*

$$\|C - g_1 g_2 \cdots g_S\| \leq \epsilon \tag{24}$$

This defines whether a gateset is a basis, or whether universal approximation is even possible (non-constructively). We will see in Chapter 3 that quantum compilers are concerned with constructive approaches to *efficiently* return such a compiled sequence $\tilde{C} = \prod g_i$. Examples of efficiency metrics are the number of returned gates S or the running time of the quantum compiling algorithm (which itself is primarily classical).

What gatesets are known to be fixed, finite, and universal, and therefore suitable bases for quantum compilation? We state without proof that the following basis satisfies these properties, and we will use it later as our circuit basis in Chapters 1 and 2.

$$\mathcal{G} = \{X, Z, H, Toffoli, CNOT\} \tag{25}$$

It is important to note that the only non-Clifford gate in the above basis is $Toffoli$. The Clifford group $\mathcal{C}_n$ by itself is *not* universal. In fact, it is provable that *any* universal gateset must possess at least one non-Clifford gate [109]. Two popular choices for the non-Clifford gate in a basis are the $T = R_Z(\pi/4)$ gate and the $Toffoli = \Lambda^2(X)$ gate. Since these are not "natively" supported (non-transversal) in many codes, they must often be implemented probabilistically using only Clifford operations and $MeasureZ$, usually by way of a so-called "magic" state. Therefore, many quantum compilers use the Clifford+T basis ($\mathcal{C}_2 \cup \{T\}$) or the Clifford+$Toffoli$ basis ($\mathcal{C}_2 \cup \{Toffoli\}$), and measure the non-Clifford gate as the most expensive resource. It is an area of active research whether T or $Toffoli$ is more efficient to implement [49, 33].

For single-qubit compilation, the $\{H, T, T^\dagger\}$ gateset is universal and plays an important role in the literature. Other compilers may add the Clifford gates S and $S^\dagger$ to the bases above, but this does not change their universality nor its asymptotic efficiency for compiling. From our discussion in this section on quantum gates and circuit bases, we are now prepared to add some additional circuit constraints to form quantum architectures.

0.2 Quantum Architectures and Models

In this dissertation, we will refer to a quantum architecture (or simply, an architecture) interchangeably with the terms *implementation* or a *circuit*, although in the latter case it is understood that the circuit has architectural constraints. We will make our definition from the introduction more formal here.

Definition 3. *A quantum architecture consists of a quantum circuit C with two main constraints.*

1. *a fault-tolerant basis (fixed, finite) basis of gates $\mathcal{G}$*

2. *a layout graph (V, E) which defines the connectivity (allowed two-qubit interactions, or edges in E) between qubits (which are nodes in V).*

The constraints are chosen to match some aspect of experimental reality. The circuit basis is usually chosen to meet fault-tolerance requirements. The layout graph is chosen to match the nearest-neighbor nature of many physical interactions.

Following Van Meter and Itoh [100], we distinguish between a model and an architectural implementation as follows. A *model* includes a set of constraints for circuit basis and layout graph, which may be common to many quantum architectures. Models allow us to abstract our study of quantum architectures further and reason about general properties of the constraints themselves. We say that an architecture is implemented *on* such-and-such a model, or that a model is meant to closely correspond to a physical technology such as superconducting qubits.

In the rest of this section, we give definitions of several important models that are common in the literature and how they are related to each other. In particular, we define

the model 2D CCNTC as a foundation for a new *hybrid* model that we introduce in Section 0.4. Finally, we define how circuit resources are counted using 2D CCNTC in the rest of this dissertation.

0.2.1 Architectural Models

The most general model is called Abstract Concurrent (AC) and allows arbitrary, long-range interactions between any qubits and concurrent operation of quantum gates on disjoint qubits. The layout graph is a complete graph with an edge between every pair of nodes. It is the model assumed in most abstract quantum algorithms. The circuit basis for this model is arbitrary (infinite) single-qubit operations and some two-qubit operation that completes it as a universal gateset (usually $CNOT$). All operations are usually counted to have unit depth and unit size.

A more specialized model restricts interactions to nearest-neighbor, two-qubit, concurrent gates (NTC). This is a submodel of AC in that it has the same circuit basis, but loosens a restriction on the layout graph, which no longer has to be complete. All AC architectures are also NTC architectures, but not vice versa.

In a more realistic restriction of NTC, we confine the qubits to lie in a regular one-dimensional chain. We call this new model 1D NTC or more commonly linear nearest-neighbor (LNN). The layout graph is now a line graph. Since Shor's original algorithm was presented in AC [87], we expect an increase in circuit resources to simulate these same long-range interaction using the nearest-neighbor interactions of NTC. This model more realistically captures multiple ion traps that are arranged in a linear chain, perhaps one where each trap encodes a single logical qubit.

An even more realistic model is described in Beckman et al. [9] called ION TRAP, which unsurprisingly, corresponds to a single linear trap where all ions are coupled by common vibrational modes [25]. The layout graph consists of a star topology, where all ion qubits are connected to the motional qubit, and the basis again consists of arbitrary single-qubit gates and fixed two qubit gates (usually $CNOT$ or the controlled phase gate $\Lambda(Z)$). All gates require a constant number of laser pulses to implement, although

some are more expensive than others. An interesting consequence of this model is that multiply-controlled single-qubit gates of the form $\Lambda^n(U)$ for $U \in U(2)$ can be performed with $O(n)$ pulses and no ancillary qubits beyond the motional qubit. Unfortunately, it is difficult to scale this configuration due to the slowing of operations for large strings of ions [43].

To relieve movement congestion up and down a linear chain, we can consider a two-dimensional regular grid (2D NTC), where each qubit has four planar neighbors, and there is an extra degree of freedom over the 1D model in which to move data. While this extra degree of freedom allows us to decrease depth over 1D NTC, it possible to improve it further. To allow this, we extend this model in three powerful yet still physically realistic ways, described in the next section.

0.2.2 The Model 2D CCNTC

Two-dimensional models are the focus of this dissertation, especially our factoring architectures in Chapters 1 and 2.

The first extension to 2D NTC allows arbitrary planar graphs with bounded degree, rather than a regular square lattice. Namely, we assume qubits lie in a plane and edges are not allowed to intersect. All qubits are accessible from above or below by control and measurement apparatus. Whereas 2D NTC conventionally assumes each qubit has four neighbors, we consider up to six neighbors in a roughly hexagonal layout. The edge length in this model is no more than twice the edge length in a regular 2D NTC lattice.

The second extension is the realistic assumption that classical control (CC) can access every qubit in parallel, and we do not count these classical resources in our implementation since they are polynomially bounded. The classical controllers correspond to fast digital computers which are available in actual experiments and are necessary for constant-depth communication in the next section.

We call an AC or NTC model augmented by these two extensions CCAC and CCNTC, respectively. We will frequently refer to NTC and CCNTC without reference to dimensionality when we are making general statements about nearest-neighbor architectures,

with or without classical controllers, regardless of the connectivity of the layout graph. Let us now formalize our model for 2D CCNTC, with definitions that are (asymptotically) equivalent to those in [82].

In our third extension, we restrict our gate set to be the fault-tolerant, finite, fixed basis $\mathcal{G}$ defined in the previous section and repeated below. Traditionally, quantum architecture papers that implement Shor's factoring algorithm have focused exclusively on nearest-neighbor constraints for the layout graph (see Section 1.3 for a full bibliography). A crucial difference in our architectures is the use of this more realistic circuit basis. We retain Rosenbaum's name 2D CCNTC, although his work does not necessarily use the same basis.

Now we define our formal model combining these three extensions.

Definition 4. *A 2D CCNTC architecture consists of*

- *a quantum computer QC which is represented by a planar graph (V, E). A node $v \in V$ represents a qubit which is acted upon in a circuit, and an undirected edge $(u, v) \in E$ represents an allowed two-qubit interaction between qubits $u, v \in V$. Each node has degree at most 6.*

- *a circuit basis $\mathcal{G} = \{X, Z, H, Toffoli, CNOT, MeasureZ\}$.*

- *a deterministic machine (classical controller) CC that applies a sequence of concurrent gates in each of D timesteps.*

- *In timestep i, CC applies a set of gates $G_i = \{g_{i,j} \in \mathcal{G}\}$. Each $g_{i,j}$ operates in one of the following two ways:*

  1. *It is a single-qubit gate from $\mathcal{G}$ acting on a single qubit $v_{i,j} \in V$*
  2. *It is the gate CNOT from $\mathcal{G}$ acting on two qubits $v_{i,j}^{(1)}, v_{i,j}^{(2)} \in V$ where $\left(v_{i,j}^{(1)}, v_{i,j}^{(2)}\right) \in E$*

  All the $g_{i,j}$ can only operate on disjoint qubits for a given timestep i. We define the support

of G_i as V_i, the set of all qubits acted upon during timestep i.

$$V_i = \bigcup_{j:g_{i,j} \in G_i} v_{i,j} \cup v_{i,j}^{(1)} \cup v_{i,j}^{(2)} \tag{26}$$

We can then define the three conventional circuit resources in this model.

circuit depth (D): the number of concurrent timesteps.

circuit size (S): the total number of (non-identity) gates applied from $\mathcal{G}$, equal to $\sum_{i=1}^{D} |G_i|$.

circuit width (W): the total number of qubits operated upon by any gate, including inputs, outputs, and ancillae. It is equal to $|\bigcup_{i=1}^{D} V_i|$.

We observe that the following relationship holds between the circuit resources. The circuit size is bounded above by the product of circuit depth and circuit width, henceforce known as the *depth-width product*. This inequality holds since in the worst case, every qubit is acted upon by a gate for every timestep of a circuit. The circuit depth is also bounded above by the size, since in the worst case, every gate is executed serially without any concurrency.

$$D \leq S \leq D \cdot W \tag{27}$$

The set $\mathcal{G}$ includes measurement in the Z basis, which is actually not a unitary operation but which may be slower than unitary operations in actual practice [29]. Therefore we count it in our resource estimates. All other gates in $\mathcal{G}$ form a universal set of unitary gates [53]. In this paper we will treat the operations in $\mathcal{G}$ as *elementary gates*, with unit depth and unit size. We also define a useful measurement in the Bell basis using operations from $\mathcal{G}$. A circuit performing this measurement is shown in Figure 2 and has depth 4, size 4, and width 2.

Counting gates from $\mathcal{G}$ as having unit size and unit depth is an overestimate compared to the models NTC and AC which are used in previous works. We intend for this

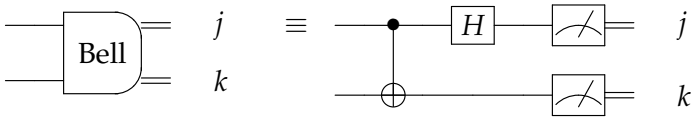

Figure 2: A circuit for measurement in the Bell state basis.

more pessimistic estimate to reflect the practical difficulties in approximating these gates using a non-Clifford gate in a fault-tolerant way, such as the T gate or the Toffoli gate [36]. However, we will still be able to use this model to achieve asymptotically superior architectures.

Next, we will show how the seemingly simple basis $\mathcal{G}$ can nevertheless allow it to simulate long-range interactions on a contiguous lattice using some communication building blocks. For each building block, including those for arithmetic in Chapter 1, we provide closed-form equations upper-bounding the required circuit resources as a function of the circuit width (input size) n. We will use the term *numerical upper bound* to distinguish these formulae from asymptotic upper bounds.

0.3 Constant-depth Communication

Communication, namely the *moving* and *copying* of quantum information, in nearest-neighbor quantum architectures is challenging. In this section we quote known results for teleportation and fanout in constant depth while also contributing a novel construction for unfanout.

The first challenge of moving quantum information from one site to another over arbitrarily long distances can be addressed by using the constant-depth teleportation circuit shown in Figure 3 due to Rosenbaum [82], illustrated using standard quantum circuit notation [76]. This requires the circuit resources shown in Table 1. The depth includes a layer of H gates; a layer of CNOTs; an interleaved layer of Bell basis measurements; and two layers of Pauli corrections (X and Z for each qubit), occurring concurrently with

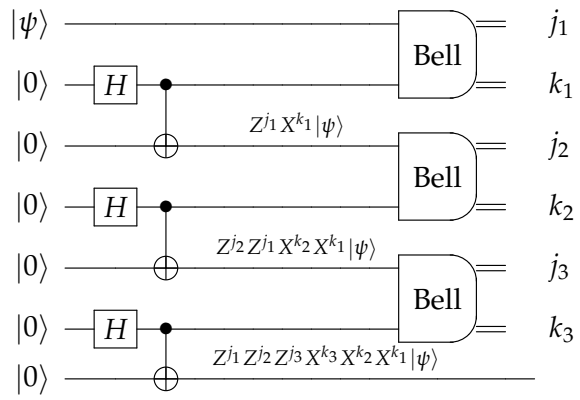

Figure 3: Constant-depth circuit based on [21, 22] for teleportation over $n = 5$ qubits [82].

resetting the $|j\rangle$ and $|k\rangle$ qubits back to $|0\rangle$. These correction layers are not shown in the circuit.

Although general cloning is impossible [76], the second challenge of copying information can be addressed by performing an unbounded quantum fanout (or just fanout) operation: $|x, y_1, \ldots, y_n\rangle \rightarrow |x, y_1 \oplus x, \ldots, y_n \oplus x\rangle$. This is used in our arithmetic circuits when a single qubit needs to control (be entangled with) a large quantum register (called a *fanout rail*). The result of a fanout is an n-qubit *cat state* stored in the fanout rail, as shown in the following equation.

$$\frac{1}{\sqrt{2}}(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}) \tag{28}$$

To implement the fanout from the basis $\mathcal{G}$, we employ a constant-depth circuit due to insight from measurement-based quantum computing [80] that first relies on the creation of an n-qubit cat state [22]. This method was communicated to us by Harrow and Fowler [45]. This circuit requires $O(1)$-depth, $O(n)$-size, and $O(n)$-width. Approximately two-thirds of the ancillae are reusable and can be reset to $|0\rangle$ after being measured. Numerical upper bounds are given in Table 1. The constant-depth fanout circuit is shown in Figure 4 for the case of fanning out a given single-qubit state $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ to four qubits.

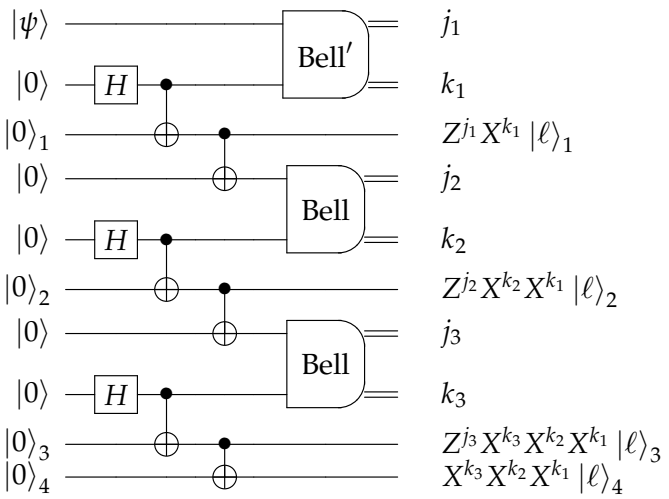

Figure 4: Constant-depth circuits based on [21, 22] for fanout [45] of one qubit to $n = 4$ entangled copies.

The technique works by creating multiple small cat states of a fixed size (in this case, three qubits), linking them together into a larger cat state of unbounded size with Bell basis measurements, and finally entangling them with the source qubit to be fanned out. The qubits marked $|\ell\rangle$ are entangled into the larger fanned out state given in Equation 29. The Pauli corrections from the cat state creation are denoted by X^{k_2}, X^{k_3}, Z^{j_2} and Z^{j_3} on qubits ending in states $|\ell\rangle_1$, $|\ell\rangle_2$, $|\ell\rangle_3$, and $|\ell\rangle_4$. The Pauli corrections X^{k_1} and Z^{j_1} are from the Bell basis measurement entangling the cat state with the source qubit (denoted Bell′).

$$Z_1^{j_1}X_1^{k_1}Z_2^{j_2}X_2^{k_2}X_2^{k_1}Z_3^{j_3}X_3^{k_3}X_3^{k_2}X_3^{k_1}X_4^{k_3}X_4^{k_2}X_4^{k_1}\left(\alpha\,|0\rangle_1\,|0\rangle_2\,|0\rangle_3\,|0\rangle_4+\beta\,|1\rangle_1\,|1\rangle_2\,|1\rangle_3\,|1\rangle_4\right) \quad (29)$$

The operators X_i^k and Z_h^j denote Pauli X and Z operators on qubits i and h, controlled by classical bits k and j, respectively. These corrections are enacted by the classical controller based on the Bell measurement outcomes (not depicted). Note the cascading nature of these corrections. There can be up to $n-1$ of these X and Z corrections on the same

qubit, which can be simplified by the classical controller to a single X and Z operation and then applied with a circuit of depth 2 and size 2. Also, given the symmetric nature of the cat state, there is an alternate set of Pauli corrections which would give the same state and is of equal size to the corrections given above.

Reversing the fanout is an operation called *unfanout*. Unfanout takes as input the n-qubit cat state from Equation 28 which is the result of a fanout. The output of unfanout, after Pauli corrections, is the product state consisting of all $|0\rangle$'s except for a single target qubit $\alpha |0\rangle + \beta |1\rangle$, which is in the same state as the original source qubit of the fanout.

In the model of [47], the fanout and unfanout were identical, elementary operations. In Figure 4, the given CCNTC fanout circuit is not its own self-inverse due to the one-way nature of the measurement and the assumption that ancillary qubits begin in the $|0\rangle$ state. Therefore, one major contribution of this current work is a CCNTC circuit which performs unfanout, or the inverse of the fanout circuit in Figure 4. The relationship of our fanout and unfanout constructions and their effect on intermediate quantum states is shown below.

$$(\alpha |0\rangle + \beta |1\rangle) \otimes |0\rangle^{\otimes n-1} \rightarrow^{fanout} \alpha |0\rangle^{\otimes n} + \beta |1\rangle^{\otimes n} \rightarrow^{unfanout} (\alpha |0\rangle + \beta |1\rangle) \otimes |0\rangle^{\otimes n-1} \quad (30)$$

We give a concrete example of our unfanout circuit in Figure 5 for $n = 7$ which executes in constant depth on 2D CCNTC. Note that the state in Equation 28 is completely symmetric in that all qubits are equivalent entangled copies of each other. Therefore, the asymmetry of the final target qubit is entirely determined by the unfanout circuit, which in this case selects the bottom qubit in the figure to be the target.

The initial fanned out state lives in a 2-dimensional subspace. The round of Hadamard gates increases its dimension to 2^n, and the two interleaved layers of $CNOT$s in a sense "disentangle" the qubits from one another, up to a Pauli Z correction. This correction, on the final target qubit, is controlled by the parity of the classical measurements on every "even" qubit (j_2 and j_4 in the figure), excluding the next-to-last qubit (j_6 in the figure). Each measurement projects the state of the target qubit into a subspace with half the dimension, so $n - 1$ measurements project the target qubit into a final 2-dimensional subspace, which is the qubit $\alpha |0\rangle + \beta |1\rangle$.

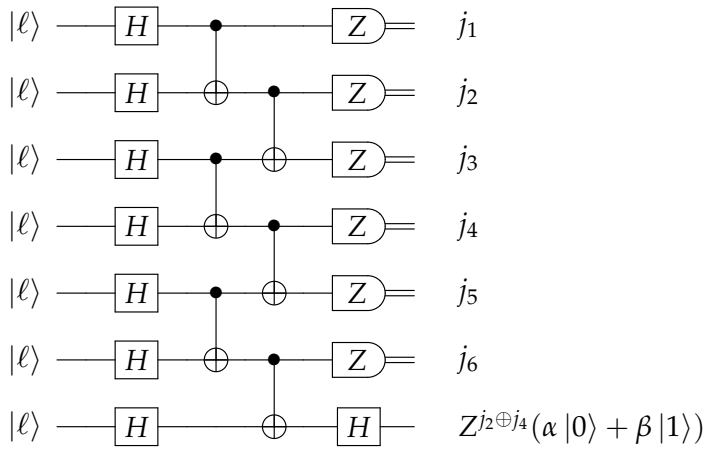

Figure 5: A novel, constant-depth circuit for unbounded quantum unfanout on CCNTC, from the 7-qubit entangled state $\alpha |0\rangle^{\otimes 7} + \beta |1\rangle^{\otimes 7}$ to the target product state $(\alpha |0\rangle + \beta |1\rangle) \otimes |0\rangle^{\otimes 6}$.

Although the circuit shown works for odd n, we can easily take into account even n with an initial *CNOT* to "uncopy" one qubit from its neighbors. The unfanout circuit in Figure 5 is the functional inverse of the fanout circuit in Figure 4, which by itself only requires a 1D layout. However, it relies on the fanned-out qubits being teleported back into adjacent positions (in constant depth), which is only possible in an overall 2D layout. This layout is not shown but can easily be constructed. The target qubit of an unfanout is usually chosen to be in the same location as the source qubit of the previous, corresponding fanout. The resources for unfanout are given in Table 1.

From an experimental perspective, it is physically efficient to create a cat state in trapped ions using the Mølmer-Sørensen gate [93][10]. However, the fanout circuit for the CCNTCM model would still be useful for other technologies, such as superconducting qubits on a 2D lattice.

Circuit Name	Depth	Size	Width
Teleportation from Figure 3	7	$3n+4$	$n+1$
Fanout from Figure 4	9	$10n-9$	$3n-1$
Unfanout from Figure 5	6	$3n+2$	n

Table 1: Circuit resources for teleportation, fanout, and unfanout.

0.4 Modules and Long-Range Interactions

Unfortunately, the model 2D CCNTC has several shortcomings which are highlighted by the constant-depth communication in the previous section. First and foremost, it is difficult to realize physically. A single, contiguous 2D lattice that contains an entire quantum architecture may be prohibitively large to manufacture or too difficult to control coherently. In practice, scalable experiments already prefer to use many smaller quantum computers that communicate by means of shared entanglement [71]. Second, for highly parallelized algorithms, the amount of quantum information undergoing active computation is so large that communicating data across the quantum processor consumes more resources than the computation itself. Finally, as quantum architects we are interested in distilling the essential locality of a computation, and distinguishing it from communication (which itself may be an interesting problem). All of these notions will be made more rigorous below, when we develop a new model to overcome these limitations.

Running an algorithm on a single 2D CCNTC architecture makes it sensitive to the geometry of the different computation sites and how they are arranged on the same lattice. Calculating the circuit resources of different geometric arrangements will have vastly different numerical constants but will asymptotically be the same. We would like a model that abstracts away this sensitivity. The key to characterizing this essential locality is the long-range interaction. Rather than simulating them using short-range interactions, we account for them as separate circuit resources. We define our new model as a *hybrid architectural model*, one which accounts for long-range and short-range interactions separately

and also constrains them according to independent graphs.

In fact, an example of this hybrid approach has already been proposed in the *quantum multicomputer* of Van Meter et al., which we abbreviate as QMC [69]. This hybrid model is also called a distributed quantum computer since it contains several communicating nodes operating in parallel. Our approach is very similar in that we divide a computation, which normally occurs on a single connected qubit graph G, into smaller subdivisions connected in a graph $\overline{G}$. If two qubits $|u\rangle$ and $|v\rangle$ are connected in G, then their two containing subdivisions $\overline{u} \ni |u\rangle$ and $\overline{v} \ni |v\rangle$ are connected in $\overline{G}$. Like QMC, we also include "flying qubits," or shared entanglement, to allow for long-range, inter-node interactions. Finally, we also regard short-range interactions between qubits in the same node as being faster and cheaper, although this is not a formal part of the model.

However, our model is different from QMC in several crucial respects. We call our circuit subdivisions *modules* to emphasize the following differences from QMC nodes.[5] Each module is a self-contained 2D CCNTC lattice, where the qubits are constrained to have nearest-neighbor interactions. Qubits in a QMC node, in contrast, are AC circuits with arbitrary connectivity. The Van Meter paper is more empirical, providing numerical calculations based on estimates of current physical technology as well as detailed schematic implementations of long-range teleportations. Our work is more abstract, providing asymptotic formulae for circuit resources. Importantly, our model allows concurrent teleportations, even among non-disjoint pairs of modules, whereas QMC nodes can only perform long-range teleportations one-at-a-time. Finally, while both QMC nodes and modules have configurable size, we have come to the conclusion that modules should contain $O(n)$ qubits each, whereas the Van Meter paper concludes that nodes have an optimal $O(1)$ size for factoring. We discuss module size in more detail in Section 0.4.3.

A related work by Beals et al. [6] describes how to achieve low-depth, generic simulations of any AC circuit, either by rearranging qubits using a sorting network on NTC or by mapping it to QMC (even with fixed-size nodes connected in a LNN topology). The depth overhead in either case for an n-qubit AC circuit is $O(\log^2 n)$. Their first approach

[5]This should not be confused with the word "modular" as in "modular arithmetic" or as referring to the modulus m that we are trying to factor.

of NTC sorting networks can be used as an alternative to hybrid architectures. As we will see later in Chapter 1, the asymptotic depth of the generic sorting network approach will be the same as our "hand-crafted" architecture for factoring, but possibly with worse constants and asymptotically larger width and size.

We have now described the motivation for a new hybrid architectural model. We have described a general feature set which allows us to capture the essential locality of a quantum algorithm while still accounting for long-range interactions.

0.4.1 The Model 2D CCNTCM

The main feature of our new submodel is that modules and long-range teleportations between them are nodes and edges, respectively, in a higher-level planar graph. The teleportations each transmit one qubit from one module to another, from any location within the source module to any location within the destination module, making use of the omnipresent classical controller.

We do not discuss the physical technology underlying the long-range teleportation other than acknowledging that it is based on shared entanglement. Namely, the two modules which are the source and destination of any long-range teleportation must each possess one half of an Einstein-Podolsky-Rosen (EPR) pair, as shown in Equation 31.

$$\frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right) \tag{31}$$

The modules can be arbitrarily far apart physically and have arbitrary connectivity. This is a reasonable assumption because entangled pairs can be generated and distributed between any two modules. This generation and distribution is counted as part of the (preprocessing) costs of the long-range teleportation itself as a one-to-one correspondence in the worst case. One could imagine that generating and distributing many entangled pairs in a batch might be more efficient physically than doing so for individual pairs. Moreover, the distribution of entangled pairs may occur along a constrained graph of module connectivity, but such constraints can be overcome with only polylogarithmic depth overhead as previously noted [6].

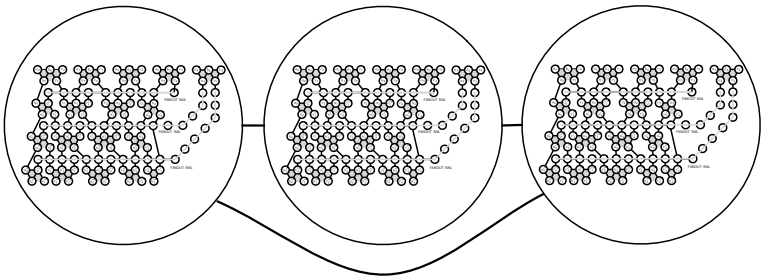

Figure 6: Three modules in the 2D CCNTCM model.

A single module can be part of multiple teleportation operations in a single timestep, as long as they involve disjoint qubits within the module. We justify this assumption in that it is possible for a quantum computer to share multiple EPR pairs with a second quantum computer, and even multiple other quantum computers, simultaneously.

We call this new model 2D CCNTCM, since it generalizes the model 2D CCNTC to have more than one module. Equivalently, each module corresponds exactly to a 2D CCNTC architecture. All modules can share a single omnipresent classical controller or a collection of multiple, inter-communicating classical controllers. A graphic depiction of three modules in 2D CCNTCM is shown in Figure 6.

Definition 5. *A 2D CCNTCM architecture consists of*

- *a quantum computer* $\overline{QC}$ *which is represented by a graph* $(\overline{V}, \overline{E})$ *of arbitrary degree. A node* $\overline{v} \in \overline{V}$ *represents a module, or a graph* (V, E) *from a 2D CCNTC architecture defined previously on page 17. An undirected edge* $(\overline{u}, \overline{v}) \in \overline{E}$ *represents an allowed teleportation from any qubit in module* $\overline{u}$ *to another qubit in module* $\overline{v}$*.*

- *a circuit basis* $\mathcal{G} = \{X, Z, H, T, T^\dagger, CNOT, MeasureZ\}$ *for the qubits* within the same *modules which is the same as for 2D CCNTC.*

- *an additional operation Teleport which only operates on qubits* in different *modules.*

- *a deterministic machine (classical controller) $\overline{CC}$ that applies a sequence of concurrent gates in each of $D + \overline{D}$ timesteps. This can be a separate classical controller for every pair of modules.*

- *In timestep i, $\overline{CC}$ applies gates $G_i = \{g_{i,j} : g_{i,j} \in \mathcal{G} \cup \{Teleport\}\}$. That is, there are two kinds of timesteps with respect to the kinds of gates which operate within them.*

  1. *In the first kind, gates are exclusively from $\mathcal{G}$, and they operate within modules as described for 2D CCNTC above. We say there are D such timesteps.*
  2. *In the second kind, gates are exclusively Teleport gates between two qubits $v_{i,j}^{(1)} \in \overline{v}_1$ and $v_{i,j}^{(2)} \in \overline{v}_2$ for (possibly non-distinct) modules $\overline{v}_1, \overline{v}_2 \in \overline{V}$. Again, all such qubits much be distinct within a timestep but the modules need not be distinct. We say there are $\overline{D}$ such timesteps.*

  Again, we define the support of G_i as V_i, the set of all qubits acted upon by any $g_{i.j}$, which includes all the modules.

$$V_i = \bigcup_{j:g_{i,j} \in G_i} v_{i,j} \cup v_{i,j}^{(1)} \cup v_{i,j}^{(2)} \tag{32}$$

We measure the efficiency of a circuit in this new module using not just the three conventional circuit resources defined for 2D CCNTC, but with three novel resources based on modules.

circuit depth (D): the depth of all operations over all modules.

circuit size (S): the total operations on all qubits within all modules.

circuit width (W): the total number of qubits, equal to the sum of all the module sizes.

module depth ($\overline{D}$): the depth of parallel teleportations of disjoint qubits between modules.

module size ($\overline{S}$): the number of total qubits teleported between any two modules over all timesteps.

module width ($\overline{W}$): the number of modules whose qubits are acted upon during any timestep.

We can define the size of a module $\overline{v} \in \overline{V}$ as the number of qubits within it, where all the module sizes add up to the total circuit width.

$$\sum_{\overline{v} \in \overline{V}} |\overline{v}| = W \tag{33}$$

0.4.2 Module Size and Model Comparisons

Modules in a 2D CCNTCM architecture do not all need to have the same size. However, it is useful to upper bound the size of any module and call this the *module size* for the entire architecture, denoted $\hat{W}$. In this dissertation, we only consider *uniformly-sized* CCNTCM architectures, ones where the module size $\hat{W}$ is upper-bounded by the average of all the module sizes, up to a constant factor. In the definition below, we iterate over all modules $\overline{v}$ in the set of all modules $\overline{V}$, where the number of modules is defined as $\overline{W} = |\overline{V}|$.

$$\hat{W} = \max_{\overline{v} \in \overline{V}} |v| \equiv O(W/\overline{W}) \tag{34}$$

We argue that module size is a key, configurable parameter of any hybrid architecture that greatly affects its resource usage. To see this, we will now compare the overhead of simulating a generic architecture (an AC circuit) on three different models with nearest-neighbor interactions: 2D NTC, 2D CCNTC, and 2D CCNTCM. This will demonstrate how 2D CCNTCM simplifies our view of computation versus communication resources.

In each of these three models, we must simulate the long-range interactions of a completely general AC circuit (with depth D, size S, and width W), using only the constraints of each particular model. We will define a depth overhead of $\tilde{D}$, a size overhead $\tilde{S}$, a width overhead $\tilde{W}$ associated purely with communication (not computation) associated with this long-range simulation. The first two models do not use modules ($\overline{W} = 1$), whereas the last model has a module size parameter $\hat{W}$ as defined above, with $1 \leq \hat{W} \leq W$ depending on the number of modules $\overline{W}$. Modules are now seen as an elegant framework

Circuit Resources	AC	2D NTC	2D CCNTC	2D CCNTCM
$D + \tilde{D}$	D	$D \cdot (1 + O(\sqrt{W})$	$D(1 + O(1))$	$D(1 + O(1))$
$S + \tilde{S}$	S	$S + D \cdot O(W)$	$S + D \cdot O(W^2)$	$S + D \cdot \overline{W} \cdot O(\hat{W}^2)$
$W + \tilde{W}$	W	W	$W(1 + W)$	$W(1 + \hat{W})$
$\overline{D}$	D			$\overline{D}$
$\overline{S}$	S			$\overline{S}$
$\overline{W}$	W			$\overline{W}$

Table 2: A comparison of simulating AC interactions on three other models. Unmarked entries signify $\overline{D} = \overline{S} = 0$ and $\overline{W} = 1$.

for interpolating our hybrid model between a completely unconstrained and a completely nearest-neighbor architectural model. When $\overline{W} = 1$, there is a contiguous lattice which corresponds exactly to 2D CCNTC. When $\overline{W} = W$, there is an unconstrained network of single qubits which corresponds exactly to AC.

Table 2 summarizes the resources used by these models to to simulate an AC circuit *with lowest depth*. In this comparison, AC is the most efficient architecture in all resources but also the most unrealistic. All other architectures pay an overhead for simulating interactions with varying degrees of nearest-neighbor realism.

A 2D NTC circuit can only use *SWAP* gates to rearrange each of the W qubits after each of D timesteps, so that all gates can occur between nearest neighbors. We choose to use a sorting network approach of Beals et al. [6] which has the following consequences. The sorting that occurs after every timestep has depth $O(\sqrt{W})$ and total communication depth $\tilde{D} = D \cdot O(\sqrt{W})$. This sorting after every timestep has size $O(W)$ (the maximum number of *SWAP*'s required) and total communication size $\tilde{S} = O(D \cdot W)$. Because sorting happens in-place, no additional ancillary qubits are needed ($\tilde{W} = 0$).

A 2D CCNTC circuit achieves lowest-depth by using the constant-depth communication of Section 0.3 and Rosenbaum's construction for reordering qubits [82]. Other, more efficient re-orderings may be possible for a specific algorithm, but the Rosenbaum reorder-

ing is completely general. This doesn't asymptotically increase the depth ($\tilde{D} = D \cdot O(1)$) but it does increase size and width by the ancillary qubits needed for the reordering grid ($\tilde{W} = W^2$, $\tilde{S} = O(W^2)$).

In the first two models, there was only one contiguous lattice ($\overline{W} = 1$) and therefore no long-range, inter-module interactions ($\overline{D} = \overline{S} = 0$). However, in the third model, we now consider multiple modules which represent a partitioning of the qubits in the original AC circuit. In the worst case, we still need to perform a Rosenbaum reordering after each of D timesteps ($\tilde{D} = D \cdot O(1)$), but this time only within smaller modules of size at most $\hat{W}$. This gives the number of communication operations $\tilde{S} = D \cdot \overline{W} \cdot O(\hat{W}^2)$.

The comparison between 2D NTC and 2D CCNTC underscores the tradeoff between a large depth on one hand and a large ancillary qubit overhead on the other hand. The comparison between 2D CCNTC and 2D CCNTCM has two noteworthy features. There is a reduction in number of intra-module operations over CCNTC if the number of modules is non-constant: $\overline{W} = \omega(1)$. However, this reduction in S must be balanced with the relative cost of $\overline{S}$, which is non-zero in CCNTCM and zero in all other models. Only physical experiments can determine the best tradeoff, but our model will facilitate such a calculation. Achieving exact bounds on $\overline{S}$ will be calculated later for factoring in Chapter 1.

0.4.3 Objections to Modules and Future Directions

Although we have demonstrate the usefulness of modules in capturing the locality of a quantum algorithm, our new CCNTCM model also contains some weaknesses. Chief among them are the possibilities of wasteful modules and the misrepresentation of the costs of long-range teleportation. We will address objections to these weaknesses here before presenting useful open questions needed to fully explore the potential of modules in hybrid architectures.

It is allowable under the CCNTCM model for an architecture to have only a small, or constant, number of qubits in each module be useful for computation. Alternatively, the qubits could be partitioned among modules such that most computationally useful

interactions must be long-range and involve qubits in other modules. Such modules are wasteful in the sense that they are not capturing any locality. We argue that such an architecture, and partitioning of qubits, can be further optimized to use fewer resources, including both long-range $\overline{S}$-type interactions and short-range S-type interactions. Even an AC circuit can have extraneous qubits and extraneous gates that are not useful for computation. Bad implementations are possible with any model, and so the wasteful module objection is not specific to CCNTCM.

Another objection to CCNTCM is that it may conceal the true costs of long-range teleportation. Whereas the models NTC and CCNTC present the communication costs of a circuit in a unified way alongside computation, it can be argued that CCNTCM is unfairly optimizing S and W compared to these two models. Indeed, for any sufficiently localized algorithm, a CCNTCM architecture can be designed with asymptotically lower S and W than a corresponding NTC or CCNTC. However, the costs of communication are not being misrepresented, they are just counted separately in the module resources $\overline{D}$, $\overline{S}$, and $\overline{W}$. Later, when physical experiments have determined the true cost of each resource, architects can make informed decisions about the best tradeoffs between module size and long-range interactions. The separate accounting of CCNTCM reflects the belief that practically, $\overline{S}$-type operations (or their equivalent simulation on a contiguous lattice) will always be much slower and expensive than S-type operations. A numerical comparison between 2D CCNTC and 2D CCNTCM for one of our factoring architectures is presented in Section 1.8.

Finally, the distribution of entangled pairs necessary to allow arbitrary connectivity of long-range teleportations has some potential pitfalls. The polylogarithmic depth overhead of EPR pair distribution is subsumed by the polylogarithmic depth of our first factoring architecture in Chapter 1, but not by our second, sublogarithmic-depth architecture in Chapter 2. These issues are dealt with in more detail in the conclusions of those chapters. More importantly, entangled pairs are likely to be generated and distributed "just-in-time" to avoid unnecessary error-correction in storing them. Therefore, this part of shared entanglement is not included as part of CCNTCM because the technology and optimal strategy for entangled pair management is still not well-developed. Physical im-

plementations could use a completely-connected optical switch, such as the one described in [71]. Furthermore, the qubits in a long-range teleportation always begin and end on the peripheries of their source and destination modules, on the edges of their respective 2D CCNTC lattices. Therefore, no additional teleportation is necessary to move those qubits into position for computation.

Beyond these objections, there is still more work to be done to fully understand the usefulness of modules and the CCNTCM model. First and foremost, the connectivity of the modules themselves. We allow teleportation between any two modules because an entangled pair can be generated and distributed between any two modules. However, in reality, this distribution might occur along constrained paths between modules. Although

Furthermore, CCNTCM can be defined for other dimensionalities, which constrain the CCNTC lattices in each modules. However, we concentrate on the 2D case in this dissertation, although the module graph $\overline{G}$ could be connected with a 1D topology independently of the modules, which could contain 2D lattices. It is a subject of future research to determine the optimal setting of connectivities of the graphs G and $\overline{G}$ independently.

Another useful investigation would be the particular tradeoff between the number of long-range interactions $\overline{S}$ and the module size $\hat{W}$. In our comparison in the previous section, we make no assumptions about the partitioning of qubits into modules or how the $\overline{S}$ long-range interactions are distributed in the circuit. This prevents us from relating $\overline{S}$, $\overline{W}$, and W in the general case. What particular features of an algorithm make this tradeoff better or worse? Unfortunately, such investigations are beyond the scope of the current work. In Chapter 1, we use the heuristic of linear size modules ($\hat{W} = O(n)$) to match the amount of computation and communication within each module. This appears to give reasonably efficient numerical results for our hybrid factoring architectures. We hope that the importance of both factoring and realistic hybrid architectures drive further work in this area.

0.5 Organization of Dissertation

This introduction has motivated the field of quantum computing in general, and quantum architecture in particular, by placing Shor's factoring algorithm in context, both historically and technologically. We have provided all the background necessary to understand and reason about bases for quantum circuits, quantum architectural models, constant-depth communication, and hybrid architectural models.

The remainder of this dissertation is organized to examine and expound upon the thesis statement above. In the first two chapters, we focus on improving the depth of factoring architectures in our hybrid 2D nearest-neighbor model using parallelization and constant-depth communication. In contrast to nearest-neighbor factoring architectures, we will refer to our results as hybrid factoring architectures, or simply hybrid factoring. Chapter 1 presents our first main result: hybrid factoring in polylogarithmic depth. In a further exponential improvement, Chapter 2 presents our second main result: hybrid factoring in sublogarithmic depth.

At this point, we have run into a seemingly fundamental limit on improving hybrid factoring to be constant depth: the compilation of arbitrary single-qubit gates to a fault-tolerant basis. We examine this *quantum compiling* limit in Chapter 3. First we present a background of quantum compiling and a survey of the current state-of-the-art in related literature to provide context. Then, in our third main result, we improve upon the Kitaev-Shen-Vyalyi procedure for generating quantum Fourier states, a vital sub-problem for single-qubit quantum compiling.

Finally, we change directions and question the approach of minimizing depth at all costs in the first two chapters. We argue that the most relevant tradeoff to consider along with decreasing depth is the increasing amount of error-correcting effort needed to maintain a useful quantum state for computation. Therefore we introduce a new resource called *circuit coherence* in Chapter 4. In this chapter, we explore circuit coherence and its relationship to other circuit resources in the hybrid model as well as to measurement-based quantum computing. We provide definitions and an algorithm for upper-bounding coherence for a given circuit. In our final main result, we show the connection between the

circuit coherence for a layered quantum circuit and a reversible pebble game, for which existing time-space tradeoffs are known. This sets the stage for proving an asymptotic separation between circuit coherence and the circuit depth-width product for the special case of factoring.

Finally we conclude our dissertation in Chapter 5 with a summary of our main results along with related open conjectures and directions for future research.

Chapter 1

SHOR'S FACTORING ALGORITHM ON A NEAREST-NEIGHBOR ARCHITECTURE

1.1 Abstract

We present a 2D nearest-neighbor quantum architecture for Shor's algorithm to factor an n-bit number in $O(\log^3 n)$ depth. Our implementation uses parallel phase estimation, constant-depth fanout and teleportation, and constant-depth carry-save modular addition. We derive upper bounds on the circuit resources of our architecture under a new 2D model which allows a classical controller and parallel, communicating modules. We provide a comparison to all previous nearest-neighbor factoring implementations. Our circuit results in an exponential improvement in nearest-neighbor circuit depth at the cost of a polynomial increase in circuit size and width.

1.2 Introduction

Shor's factoring algorithm is a central result in quantum computing, with an exponential speed-up over the best known classical algorithm [87]. As the most notable example of a quantum-classical complexity separation, much effort has been devoted to implementations of factoring on a realistic architectural model of a quantum computer [8, 59, 99, 100, 101]. We can bridge the gap between the theoretical algorithm and a physical implementation by describing the layout and interactions of qubits at an intermediate, architectural level of abstraction. This gives us a model for measuring circuit resources and their tradeoffs. In this work, we present a circuit implementation for prime factorization of an n-bit integer on a two-dimensional architecture that allows concurrent (parallel) two-qubit operations between neighboring qubits, an omnipresent classical controller, and modules which are allowed to teleport qubits to each other. We call this new model 2D CCNTCM. We show that our circuit construction is asymptotically more efficient in circuit

depth than previous state-of-the-art techniques for nearest-neighbor architectures, achieving a depth of $O(\log^3 n)$, a size of $O(n^4 \log n)$, and a width of $O(n^4)$ qubits, as detailed in Table 1.2 of Section 1.8.

Our technique hinges on several key building blocks, including constant-depth communication from Section 0.3 and hybrid architectures with modules from Section 0.4. Section 1.3 places our work in the context of existing results. In Section 1.4, we provide a self-contained pedagogical review of the carry-save technique and encoding. In Section 1.5 we modify and extend the carry-save technique to a 2D modular adder, which we then use as a basis for a modular multiplier (Section 1.6) and a modular exponentiator (Section 1.7). For each building block, we provide numerical upper bounds for the required circuit resources. Finally, we compare our asymptotic circuit resource usage with other factoring implementations.

1.3 Related Work

Our work builds upon ideas in classical digital and reversible logic and their extension to quantum logic. Any circuit implementation for Shor's algorithm requires a quantum adder. Gossett proposed a quantum algorithm for addition using classical carry-save techniques to add in constant-depth and multiply in logarithmic-depth, with a quadratic cost in qubits (circuit width) [41]. The techniques relies on encoded addition, sometimes called a $3 \to 2$ adder, and derives from classical Wallace trees [104].

Takahashi and Kunihiro discovered a linear-depth and linear-size quantum adder using zero ancillae [95]. They also developed an adder with tradeoffs between $O(n/d(n))$ ancillae and $O(d(n))$-depth for $d(n) = \Omega(\log n)$ [98]. Their approach assumes unbounded fanout, which had not previously been mapped to a nearest-neighbor circuit until our present work.

Studies of architectural constraints, namely restriction to a 2D planar layout, were experimentally motivated. For example, these layouts correspond to early ion trap proposals [52] and were later analyzed at the level of physical qubits and error correction in the context of Shor's algorithm [106]. Choi and Van Meter designed one of the first

adders targeted to a 2D architecture and showed it runs in $\Theta(\sqrt{n})$-depth on 2D NTC [24] using $O(n)$-qubits with dedicated, special-purpose areas of a physical circuit layout.

Modular exponentiation is a key component of quantum period-finding (QPF), and its efficiency relies on that of its underlying adder implementation. Since Shor's algorithm is a probabilistic algorithm, multiple rounds of QPF are required to amplify success probability arbitrarily close to 1. It suffices to determine the resources required for a single round of QPF with a fixed, modest success probability (in the current work, 3/4).

The most common approach to QPF performs controlled modular exponentiation followed by an inverse quantum Fourier transform (QFT) [76]. We will call this *serial QPF*, which is used by the following implementations.

Beauregard [8] constructs a cubic-depth quantum period-finder using only $2n + 3$ qubits on AC. It combines the ideas of Draper's transform adder [30], Vedral et al.'s modular arithmetic blocks [103], and a semi-classical QFT. This approach was subsequently adapted to 1D NTC by Fowler, Devitt, and Hollenberg [37] to achieve resource counts for an $O(n^3)$-depth quantum period-finder. Kutin [59] later improved this using an idea from Zalka for approximate multipliers to produce a QPF circuit on 1D NTC in $O(n^2)$-depth. Thus, there is only a constant overhead from Zalka's own factoring implementation on AC, which also has quadratic depth [108]. Takahashi and Kunihiro extended their earlier $O(n)$-depth adder to a factoring circuit in $O(n^3)$-depth with linear width [96]. Van Meter and Itoh explore many different approaches for serial QPF, with their lowest achievable depth being $O(n^2 \log n)$ with $O(n^2)$ on NTC [100]. Cleve and Watrous calculate a factoring circuit depth of $O(\log^3 n)$ and corresponding circuit size of $O(n^3)$ on AC, assuming an adder which has depth $O(\log n)$ and $O(n)$ size and width. We beat this depth and provide a concrete architectural implementation using an adder with $O(1)$-depth and $O(n)$ size and width.

In the current work, we assume that errors do not affect the storage of qubits during the circuit's operation. An alternate approach is taken by Miquel [70] and Garcia-Mata [38], who both numerically simulate Shor's algorithm for factoring specific numbers to determine its sensitivity to errors. Beckman et al. provide a concrete factoring implementation in ion traps with $O(n^3)$ depth and size and $O(n)$ width [9].

In all the previous works, it is assumed that qubits are expensive (width) and that execution time (depth) is not the limiting constraint. We make the alternative assumption that ancillae are cheap and that fast classical control is available which allows access to all qubits in parallel. Therefore, we optimize circuit depth at the expense of width. We compare our work primarily to Kutin's method [59].

These works also rely on serial QPF which in turn relies on an inverse QFT. On an AC architecture, even when approximating the (inverse) QFT by truncating two-qubit $\pi/2^k$ rotations beyond $k = O(\log n)$, the depth is $O(n \log n)$ to factor an n-bit number. To be implemented fault-tolerantly on a quantum device, rotations in the QFT must then be compiled into a discrete gate basis. This requires at least a $O(\log(1/\epsilon))$ overhead in depth to approximate a rotation with precision ϵ [46, 53]. We would like to avoid the use of a QFT due to its compilation overhead.

There is an alternative, parallel version of phase estimation [26, 53], which we call *parallel QPF* (we refer the reader to Section 13 of [53] for details), which decreases depth in exchange for increased width and additional classical post-processing. This eliminates the need to do an inverse QFT. We develop a nearest-neighbor factoring circuit based on parallel QPF and our proposed 2D quantum arithmetic circuits. We show that it is asymptotically more efficient than the serial QPF method. We compare the circuit resources required by our work with existing serial QPF implementations in Table 1.2 of Section 1.8. However, a recent result by [48] allows one to enact a QFT using only Clifford gates and a Toffoli gate in $O(\log^2 n)$ expected depth. This would allow us to greatly improve the constants in our circuit resource upper bounds in Section 1.7 by combining a QFT with parallel multiplication similar to the approach described in [100, 26].

We also note that recent results by Browne, Kashefi, and Perdrix (BKP) connect the power of measurement-based quantum computing to the quantum circuit model augmented with unbounded fanout [22]. Their model, which we adapt and call CCNTC, uses the classical controller mentioned in Section 0.3. Using results by Høyer and Špalek [47] that unbounded quantum fanout would allow for a constant-depth factoring algorithm, they conclude that a probabilistic polytime classical machine with access to a constant-depth one-way quantum computer would also be able to factor efficiently.

1.4 *The Constant-Depth Carry-Save Technique*

Our 2D factoring approach rests on the central technique of the constant-depth carry-save adder (CSA) [41], which converts the sum of three numbers a, b, and c, to the sum of two numbers u and v: $a + b + c = u + v$. The explanation of this technique and how it achieves constant depth requires the following definitions.

A *conventional number* x can be represented in n bits as $x = \sum_{i=0}^{n-1} 2^i x_i$, where $x_i \in \{0, 1\}$ denotes the i-th bit of x, which we call an i-bit and has significance 2^i, and the 0-th bit is the low-order bit.[1] Equivalently, x can be represented as a (non-unique) sum of two smaller, $(n-1)$-bit, conventional numbers, u and v. We say $(u + v)$ is a *carry-save encoded*, or CSE, number. The CSE representation of an n-bit conventional number consists of $2n - 2$ individual bits where v_0 is always 0 by convention.

Consider a CSA operating on three bits instead of three numbers; then a CSA converts the sum of three i-bits into the sum of an i-bit (the *sum* bit) and an $(i+1)$-bit (the *carry* bit): $a_i + b_i + c_i = u_i + v_{i+1}$. By convention, the bit u_i is the parity of the input bits ($u_i = a_i \oplus b_i \oplus c_i$) and the bit v_{i+1} is the majority of $\{a_i, b_i, c_i\}$. Figure 1.1 gives a concrete example, where $(u + v)$ has $2n - 2 = 8$ bits, not counting v_0.

It will also be useful to refer to a subset of the bits in a conventional number using subscripts to indicate a range of indices:

$$x_{(j,k)} \equiv \sum_{i=j}^{k} 2^i x_i \qquad x_{(i)} \equiv x_{(i,i)} = 2^i x_i. \tag{1.1}$$

Using this notation, the following identity holds:

$$x_{(j,k)} = x_{(j,\ell)} + x_{(\ell+1,k)}, \qquad \text{for all } j \le \ell < k. \tag{1.2}$$

We can express the relationship between the bits of x and $(u + v)$ as follows:

$$x = x_{(0,n-1)} \equiv u + v = u_{(0,n-2)} + v_{(1,n-1)}. \tag{1.3}$$

Finally, we denote arithmetic modulo m with square brackets.

$$x_{(j,k)} \bmod m = x_{(j,k)}[m] \tag{1.4}$$

[1] It will be clear from the context whether we mean an i-bit, which has significance 2^i, or an i-bit number.

$$x = 30 = u + v = 8 + 22 = \left\{ \begin{array}{ccccc} & u_3 & u_2 & u_1 & u_0 \\ v_4 & v_3 & v_2 & v_1 & \\ \hline x_4 & x_3 & x_2 & x_1 & x_0 \end{array} \right\} = \left\{ \begin{array}{ccccc} & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & \\ \hline 1 & 1 & 1 & 1 & 0 \end{array} \right\}$$

Figure 1.1: An example of carry-save encoding for the 5-bit conventional number 30.

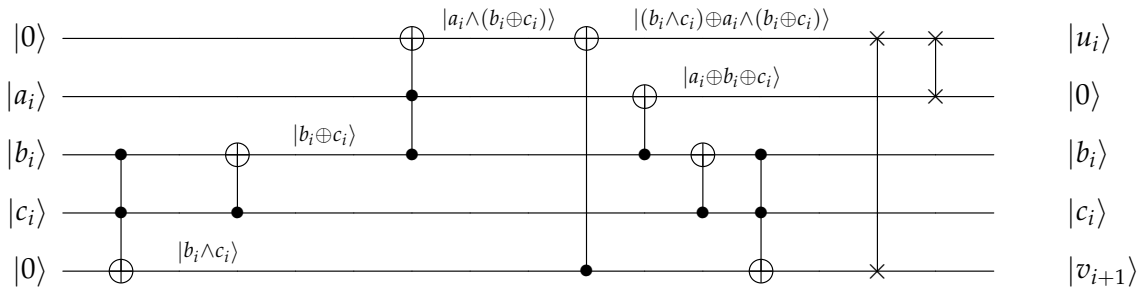

Figure 1.2: Carry-save adder circuit for a single bit position i: $a_i + b_i + c_i = u_i + v_{i+1}$. The Toffoli gate is further decomposed into the circuit of Figure 1.3.

Figure 1.2 gives a circuit description of carry-save addition (CSA) for a single bit position i. The resources for this circuit are given in Table 1.1, using the resources for the Toffoli gate (in the same table) based on [4]. We note here that a more efficient decomposition for the Toffoli is possible using a distillation approach described in [49].

We must lay out the circuit to satisfy a 2D NTC model. The Toffoli gate decomposition in [4], duplicated in Figure 1.3, requires two control qubits and a single target qubit to be mutually connected to each other. Given this constraint, and the interaction of the CNOTs in Figure 1.2, we can rearrange these qubits on a 2D planar grid and obtain the layout shown in Figure 1.4, which satisfies our 2D NTC model. Qubits $|a\rangle_i$, $|b\rangle_i$, and $|c\rangle_i$ reside at the top of Figure 1.4, while qubits $|u_i\rangle$ and $|v_{i+1}\rangle$ are initialized to $|0\rangle$. Upon completion of the circuit, qubit $|a_i\rangle$ is in state $|0\rangle$, as seen from the output in Figure 1.2.

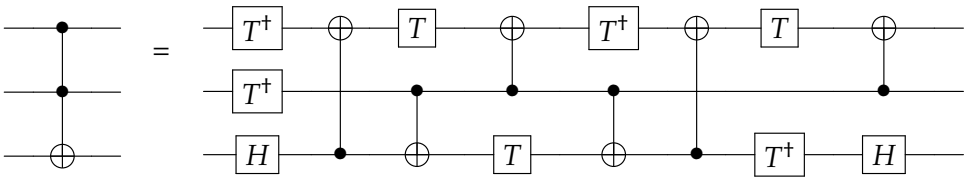

Figure 1.3: The depth-efficient Toffoli gate decomposition from [4].

Circuit Name	Depth	Size	Width
Toffoli gate from [4] and Figure 1.3	8	15	3
Single-bit 3-to-2 adder from Figure 1.2	33	55	5

Table 1.1: Circuit resources for Toffoli and single-bit addition.

Note that this construction uses more gates and one more ancilla than the equivalent quantum full adder circuit in Figure 5 of [41]. However this is necessary in order to meet our architectural constraints and does not change the asymptotic results. Also in Figure 1.4 is a variation called a $2 \to 2$ adder, which simply re-encodes two i-bits into an i-bit and an $(i+1)$-bit. The $2 \to 2$ adder uses at most the resources of a $3 \to 2$ adder, so we can count it as such in our calculations. It will be useful in the next section.

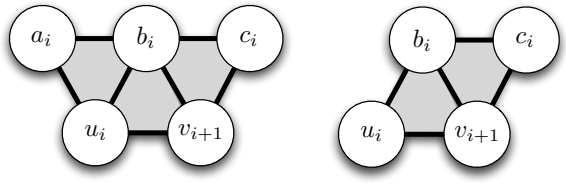

Figure 1.4: The carry-save adder (CSA), or $3 \to 2$ adder, and carry-save $2 \to 2$ adder.

At the level of numbers, the sum of three n-bit numbers can be converted into the sum of two n-bit numbers by applying a *CSA layer* of n parallel, single-bit CSA circuits (Fig. 1.2). Since each CSA operates in constant depth, the entire layer also operates in constant depth, and we have achieved (non-modular) addition. Each single addition of three n-bit numbers requires $O(n)$ circuit width.

1.5 Quantum Modular Addition

To perform addition of two numbers a and b modulo m, we consider the variant problem of modular addition of three numbers to two numbers: Given three n-bit input numbers a, b, and c, and an n-bit modulus m, compute $(u + v) = (a + b + c)[m]$, where $(u + v)$ is a CSE number.

In this section, we provide an alternate, pedagogical explanation of Gossett's modular reduction [41]. Later, we contribute a mapping of this adder to a 2D architecture, using unbounded fanout to maintain constant depth for adding back modular residues. This last step is absent in Gossett's original approach.

To start, we will demonstrate the basic method of modular addition and reduction on an n-bit conventional number. In general, adding two n-bit conventional numbers will produce an overflow bit of significance 2^n, which we can truncate as long as we add back its modular residue $2^n \bmod m$. How can we guarantee that we won't generate another overflow bit by adding back the modular residue? It turns out we can accomplish this by allowing a slightly larger input and output number ($n + 1$ bits in this case), truncating multiple overflow bits, and adding back their n-bit modular residues.

For two $(n + 1)$-bit conventional numbers x and y, we truncate the three high-order bits of their sum $z_{n-1,n+3}$ and add back their modular residue $x_{(n-1,n)}[m]$:

$$\begin{aligned} x + y \bmod m &= z_{(0,n+1)}[m] \\ &= z_{(0,n-2)} + z_{(n-1,n+1)}[m]. \end{aligned} \tag{1.5}$$

Since both the truncated number $z_{(0,n-2)}$ and the modular residue are n-bit numbers, their sum is an $(n + 1)$-bit number as desired, equivalent to $x[m]$.

Now we must do the same modular reduction on a CSE number $(u+v)$, which in this case represents an $(n+2)$-bit conventional number and has $2n+3$ bits. First, we truncate the three high-order bits (v_n, u_{n-1}, v_{n-1}) of $(u+v)$, yielding an n-bit conventional number with a CSE representation of $2n$ bits: $\{u_0, u_1, \ldots, u_{n-1}\} \cup \{v_1, v_2, \ldots, v_{n-1}\}$. Then we add back the three modular residues $(v_{(n+1)}[m], u_{(n)}[m], v_{(n)}[m])$, and we are guaranteed not to generate additional overflow bits (of significance 2^n or higher). This equivalence is shown in Equation 1.6.

$$\begin{aligned}(u+v)[m] &= \left(u_{(0,n+1)} + v_{(1,n+2)}\right)[m] \\ &= u_{(0,n)} + v_{(1,n)} + \\ &\quad u_{(n+1)}[m] + v_{(n+1)}[m] + v_{(n+2)}[m] \end{aligned} \tag{1.6}$$

Lemma 1 (Modular Reduction in Constant Depth)**.** *The modular addition of three n-bit numbers to two n-bit numbers can be accomplished in constant depth with $O(n)$ width in 2D CCNTC.*

Proof. Our goal is to show how to perform modular addition while keeping our numbers of a fixed size by treating overflow bits correctly. We map the proof of [41] to 2D CCNTC and show that we meet our required depth and width. First, we enlarge our registers to allow the addition of $(n+2)$-bit numbers, while keeping our modulus of size n bits. (In Gossett's original approach, he takes the equivalent step of restricting the modulus to be of size $(n-2)$ bits.) We accomplish the modular addition by first performing a layer of non-modular addition, truncating the three high-order overflow bits, and then adding back modular residues controlled on these bits in three successive layers, where we are guaranteed that no additional overflow bits are generated in each layer. This is illustrated for a 3-bit modulus and 5-bit registers in Figure 1.5.

We use the following notation. The non-modular sum of the first layer is u and v. The CSE output of the first modular reduction layer is u' and v', and the modular residue is written as $c^{v_{n+1}}$ to mean the precomputed value 2^{n+1} mod m controlled on v_{n+1}. The CSE output of the second modular reduction layer is u'' and v'', and the modular residue

	a_4	a_3	a_2	a_1	a_0	5-bit input number a	
	b_4	b_3	b_2	b_1	b_0	5-bit input number b	
	c_4	c_3	c_2	c_1	c_0	5-bit input number c	[Layer 1]
	u_4	u_3	u_2	u_1	u_0	truncate u_4	
v_5	v_4	v_3	v_2	v_1		truncate v_4, v_5	
			$c_2^{v_4}$	$c_1^{v_4}$	$c_0^{v_4}$	add back 2^4 mod m controlled on v_4	[Layer 2]
		u'_3	u'_2	u'_1	u'_0		
	v'_4	v'_3	v'_2	v'_1			
			$c_2^{u_4}$	$c_1^{u_4}$	$c_0^{u_4}$	add back 2^4 mod m controlled on u_4	[Layer 3]
	u''_4	u''_3	u''_2	u''_1	u''_0	the bit u''_4 is the same as v'_4	
	v''_4	v''_3	v''_2	v''_1			
			$c_2^{v_5}$	$c_1^{v_5}$	$c_0^{v_5}$	add back 2^5 mod m controlled on v_5	[Layer 4]
	u'''_4	u'''_3	u'''_2	u'''_1	u'''_0	Final CSE output with 5 bits	
	v'''_4	v'''_3	v'''_2	v'''_1		Final CSE output with 5 bits	

Figure 1.5: A schematic proof of Gossett's constant-depth modular reduction for $n = 3$.

is written as $c^{u_{n+1}}$ to mean the precomputed value 2^{n+1} mod m controlled on u_{n+1}. The CSE output of the third and final modular reduction layer is u''' and v''', and the modular residue is written as $c^{v_{n+2}}$ to mean the precomputed value 2^{n+2} mod m controlled on v_{n+2}.

We show that no layer generates an overflow $(n+2)$-bit, namely in the v component of any CSE output. (The u component will never exceed the size of the input numbers.)

First, we know that no v'_{n+2} bit is generated after the first modular reduction layer, because we have truncated away all $(n+1)$-bits. Second, we know that no v''_{n+2} bit is generated because we only have one $(n+1)$-bit to add, v'_{n+1}. Finally, we need to show that $v'''_{n+2} = 0$ in the third modular reduction layer.

Since $u'_{(n)} + v'_{(n+1)} = u_{(n)} + v_{(n)} \leq 2^{n+1}$, the bits u'_n and v'_{n+1} cannot both be 1. But $u''_{n+1} = v'_{n+1}$ and $v''_{n+1} = u'_n \wedge v'_n$, so u''_{n+1} and v''_{n+1} cannot both be 1, and hence $v'''_{n+2} = 0$. Everywhere we use the fact that the modular residues are restricted to n bits. Therefore, the modular sum is computed as the sum of two $(n+2)$-bit numbers with no overflows in constant-depth. □

As a side note, we can perform modular reduction in one layer instead of three by decoding the three overflow bits into one of seven different modular residues. This can also be done in constant depth, and in this case we only need to enlarge all our registers to $(n+1)$ bits instead of $(n+2)$ as in the proof above. We omit the proof for brevity.

In the following two subsections, we give a concrete example to illustrate the modular addition circuit as well as a numerical upper bound for the general circuit resources.

1.5.1 A Concrete Example of Modular Addition

A 2D CCNTC circuit for modular addition of 5-bit numbers using four layers of parallel CSA's is shown graphically in Figure 1.6 which corresponds directly to the schematic proof in Figure 1.5. Note that in Figure 1.6, the least significant qubits are on the left, and in Figure 1.5, the least significant qubits are on the right. Figure 1.6 also represents the approximate physical layout of the qubits as they would look if this circuit were to be fabricated. Here, we convert the sum of three 5-bit integers into the modular sum of two 5-bit integers, with a 3-bit modulus m. In the first layer, we perform 4 CSA's in parallel on the input numbers (a, b, c) and produce the output numbers (u, v).

As described above, we truncate the three high-order bits during the initial CSA round (bits u_4, v_4, v_5) to retain a 4-bit number. Each of these bits serves as a control for adding

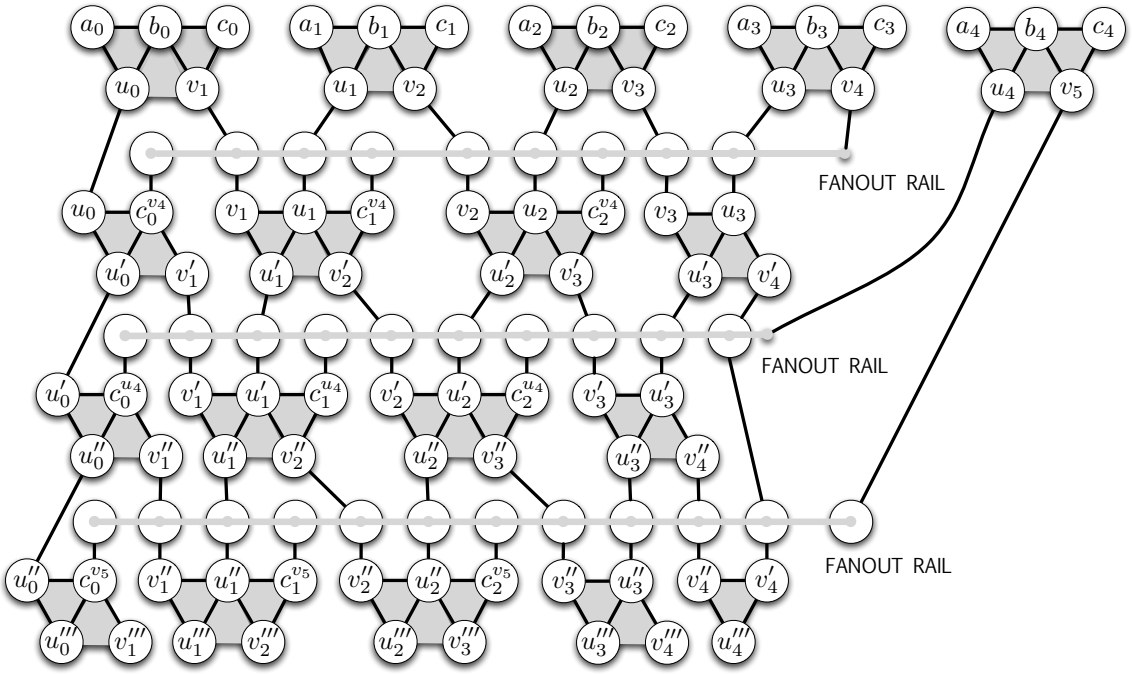

Figure 1.6: Addition and three rounds of modular reduction for a 3-bit modulus.

its modular residue to a running total. We can classically precompute $2^4[m]$ for the two additions controlled on u_4 and v_4 and $2^5[m]$ for the addition controlled on v_5.

In Layer 2, we use a constant-depth fanout rail (see Figure 4) to distribute the control bit v_4 to its modular residue, which we denote as $|c^{v_4}\rangle \equiv \left|2^4[m] \cdot v_4\right\rangle$. The register c^{v_4} has n bits, which we add to the CSE results of layer 1. The results u_i and v_{i+1} are teleported into layer 3. The exception is v'_4 which is teleported into layer 4, since there are no other 4-bits to which it can be added. Wherever there are only two bits of the same significance, we use the 2-2 adder from Section 1.4.

Layer 3 operates similarly to layer 2, except that the modular residue is controlled on u_4: $|c^{u_4}\rangle \equiv \left|2^4[m] \cdot u_4\right\rangle$. The register c^{u_4} has 3 bits, which we add to the CSE results of layer 2, where u'_i and v'_{i+1} are teleported forward into layer 4.

Layer 4 is similar to layers 2 and 3, with the modular residue controlled on v_5: $|c^{v_5}\rangle \equiv$

$\left|2^5[m] \cdot v_5\right\rangle$. The register c^{v_5} has 3 bits, which we add to the CSE results of layer 3. There is no overflow bit v_5''', and no carry bit from v_4'' and v_4' as argued in the proof of Lemma 1. The final modular sum $(a+b+c)[m]$ is $u''' + v'''$.

The general circuit for adding three n-qubit quantum integers to two n-qubit quantum integers is called a *CSA tile*. Each CSA tile in our architecture corresponds to its own module, and it will be represented by the symbol in Figure 1.7 for the rest of this paper. We call this an n-bit modular adder, even though it accepts $(n+2)$-bit inputs, because the size of the modulus is still n bits.

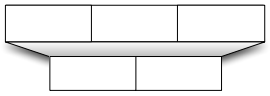

Figure 1.7: Symbol for an n-bit 3-to-2 modular adder, also called a CSA tile.

1.5.2 *Quantum Circuit Resources for Modular Addition*

We now calculate numerical upper bounds for the circuit resources of the n-bit 3-to-2 modular adder described in the previous section. There are four layers of non-modular n'-bit 3-to-2 adders, each of which consists of n' parallel single-bit adders whose resources are detailed in Table 1.1. For factoring an n-bit modulus, we have $n' = n+2$ in the first and fourth layers and $n' = n+1$ in the second and third layers.

After each of the first three layers, we must move the output qubits across the fanout rail to be the inputs of the next layer. We use two swap gates, which have a depth and size of 6 CNOTs each, since the depth of teleportation is only more efficient for moving more than two qubits. The control bit for each modular residue needs to be teleported 0, 4, and 7 qubits respectively according to the diagram in Figure 1.6, before being fanned out n times along the fanout rails, where the fanned out copies will end up in the correct position to be added as inputs.

The resources for the n-bit 3-to-2 modular adder depicted in Figure 1.6 are given below. The formulae reflect the resources needed for both computing the output in the forward direction (including creating an entangled fanned-out state controlled on overflow qubits) and also uncomputing ancillae in the backward direction (including disentangling previous fanned-out copies).

The circuit depth is $O(1)$:

$$374. \tag{1.7}$$

The circuit size is $O(n)$:

$$551n + 757. \tag{1.8}$$

The circuit width is $O(n)$:

$$33n + 47. \tag{1.9}$$

1.6 Quantum Modular Multiplication

We can build upon our carry-save adder to implement quantum modular multiplication in logarithmic depth. We start with a completely classical problem to illustrate the principle of multiplication by repeated addition. Then we consider modular multiplication of two quantum integers in a serial and a parallel fashion in Section 1.6.1. Both of these problems use as subroutines *partial product creation*, which we define and solve in Section 1.6.2 and *modular multiple addition*, which we define and solve in Section 1.6.3.

First we consider a completely classical problem: given three n-bit classical numbers a, b, and m, compute $c = ab \bmod m$, where c is allowed to be in CSE.

We only have to add shifted multiples of a to itself, "controlled" on the bits of b. There are n shifted multiples of a, let's call them $z^{(i)}$, one for every bit of b: $z^{(i)} = 2^i a b_i \bmod m$. We can parallelize the addition of n numbers in a logarithmic depth binary tree to get a total depth of $O(\log n)$.

1.6.1 Modular Multiplication of Two Quantum Integers

We now consider the problem of multiplying a classical number controlled on a quantum bit with a *quantum integer*[2], which is a quantum superposition of classical numbers:

> Given an n-qubit quantum integer $|x\rangle$, a control qubit $|p\rangle$, and two n-bit classical numbers a and m, compute $|c\rangle = |xa[m]\rangle$, where c is allowed to be in CSE.

This problem occurs naturally in modular exponentiation (described in the next section) and can be considered *serial multiplication*, in that t quantum integers are multiplied in series to a single quantum register. This is used in serial QPF as mentioned in Section 1.3.

We first create n quantum integers $\left|z^{(i)}\right\rangle$, which are shifted multiples of the classical number a controlled on the bits of x: $\left|z^{(i)}\right\rangle \equiv \left|2^i a[m] \cdot x_i\right\rangle$. These are typically called *partial products* in a classical multiplier. How do we create these numbers, and what is the depth of the procedure? First, note that $\left|2^i a[m]\right\rangle$ is a classical number, so we can precompute them classically and prepare them in parallel using single-qubit operations on n registers, each consisting of n ancillae qubits. Each n-qubit register will hold a future $\left|z^{(i)}\right\rangle$ value. We then fan out each of the n bits of x, n times each, using an unbounded fanout operation so that n copies of each bit $|x_i\rangle$ are next to register $\left|z^{(i)}\right\rangle$. This takes a total of $O(n^2)$ parallel CNOT operations. We then entangle each $\left|z^{(i)}\right\rangle$ with the corresponding x_i. After this, we interleave these numbers into groups of three using constant-depth teleportation. This reduces to the task of modular multiple addition in order to add these numbers down to a single (CSE) number modulo m, which is described in Section 1.6.3.

Finally, we tackle the most interesting problem:

> Given two n-qubit quantum integers $|x\rangle$ and $|y\rangle$ and an n-bit classical number m, compute $|c\rangle = |xy \bmod m\rangle$, where $|c\rangle$ is allowed to be in CSE.

---

[2]In this paper, an n-qubit quantum integer is a general superposition of up to 2^n classical integers. As a special case, a classical number controlled on a single qubit is a superposition of 2 classical integers.

This can be considered *parallel multiplication* and is responsible for our logarithmic speedup in modular exponentiation and parallel QPF.

Instead of creating n quantum integers $\left|z^{(i)}\right\rangle$, we must create up to n^2 numbers $\left|z^{i,j}\right\rangle$ for all possible pairs of quantum bits x_i and y_j, $i,j \in \{0,\ldots,n-1\}$: $\left|z^{i,j}\right\rangle \equiv \left|2^i 2^j [m] \cdot x_i \cdot y_j\right\rangle$. We create these numbers using a similar procedure to the previous problem. Adding n^2 quantum integers of n qubits each takes depth $O(\log(n^2))$, which is still $O(\log n)$. Creating $n^2 \times n$-bit quantum integers takes width $O(n^3)$. Numerical constants are given for these resource estimates in Section 1.6.4 for the entire modular multiplier.

Here is an outline of our modular multiplier construction, combining the two halves of partial product creation (Section 1.6.2) and modular multiple addition (Section 1.6.3).

1. Initially, the inputs consist of the CSE quantum integers x and y, each with $2n+3$ bits, sitting on adjacent edges of a square lattice that has sides of length $3(2n+3)$ qubits.

2. For each of $\lceil \log_2(2n+3) \rceil$ rounds:

   (a) Of the existing $\{x_i\}$ and $\{y_j\}$ bits, apply a CNOT to create an entangled copy in an adjacent qubit.

   (b) Teleport this new copy halfway between its current location and the new copy.

   (c) At every site where an $|x_i\rangle$ and an $\left|y_j\right\rangle$ meet, apply a Toffoli gate to create $\left|x_i \cdot y_j\right\rangle$.

   (d) Teleport $\left|x_i \cdot y_j\right\rangle$ to the correct z-site module.

3. Within each z-site module, fanout $\left|x_i \cdot y_j\right\rangle$ up to n times, corresponding to each 1 in the modular residue $2^i 2^j$ mod m, to create the n-qubit quantum integer $\left|z^{(i,j)}\right\rangle$.

4. For each triplet of z-site modules, teleport the quantum integers $\left|z^{(i,j)}\right\rangle$ to a CSA tile module, interleaving the three numbers so that bits of the same significance are adjacent. This concludes partial product creation (Section 1.6.2).

5. Perform modular multiple addition (described in Section 1.6.3) on t' n-qubit quantum integers down to 2 n-qubit quantum integers (one CSE number).

6. Uncompute all the previous steps to restore ancillae to $|0\rangle$.

1.6.2 *Partial Product Creation*

This subroutine describes the procedure of creating $t' = O(n^2)$ partial products of the CSE quantum integers x and y, each with $2n+3$ bits each. We will now discuss only the case of parallel multiplication. Although we will not provide an explicit circuit for this subroutine, we will outline our particular construction and give a numerical upper bound on the resources required.

First, we need to generate the product bits $|x_i \cdot y_j\rangle$ for all possible $(2n+3)^2$ pairs of $|x_i\rangle$ and $|y_j\rangle$. A particular product bit $|x_i \cdot y_j\rangle$ controls a particular classical number, the n-bit modular residue $2^i 2^j [m]$, to form the partial product $\left|z^{(i,j)}\right\rangle$ defined in the previous section. However, some of these partial products consist of only a single qubit, if $2^i 2^j < 2^n$, which is the minimum value for an n-bit modulus m. There are at least $2n^2 - 2n + 1$ such single-bit partial products, which can be grouped into at most $(2n+3) \times n$-bit numbers. Of the $(2n+3)^2$ possible partial products, this leaves the number of remaining n-bit partial products as at most $2n^2 + 14n + 8$. Therefore we have a maximum number of n-bit partial products, which we will simply refer to as t' from now on.

$$t' = 2n^2 + 16n + 11 \tag{1.10}$$

The creation of the product bits $|x_i \cdot y_j\rangle$ occurs on a square lattice of $(3(2n+3))^2$ qubits, with the numbers $|x_i\rangle$ and $|y_j\rangle$ located on adjacent edges. The factor of 3 in the size of the lattice allows the $|x_i\rangle$ and $|y_j\rangle$ bits move past each other. The $|x_i\rangle$ bits are teleported along an axis that is perpendicular to the teleportation axis for the $|y_j\rangle$ bits, and vice versa. Product bit creation, and this square lattice, comprise a single module. In $\lceil \log_2(2n+3) \rceil$ rounds, these bits are copied via a CNOT and teleported to the middle of a recursively halved interval of the grid. The copied bits $|x_i\rangle$ and $|y_j\rangle$ first form 1 line,

then 3 lines, then 7 lines, and so forth, intersecting at 1 site, then 9 sites, then 49 sites, and so forth. There are $\lceil \log_2(2n+3) \rceil$ such rounds.

At each intersection, a Toffoli gate is used to create $\left|x_i \cdot y_j\right\rangle$ from the given $|x_i\rangle$ and $\left|y_j\right\rangle$. These product bits are then teleported away from this qubit, out of this product bit module, to different modules where the $\left|z^{(i,j)}\right\rangle$ numbers are later generated, called z-sites. There are t' z-site modules which each contain an n-qubit quantum integer. Any round of partial product generation will produce at most as many product bits $x_i \cdot y_j$ as in the last round, which is half the total number of $(2n+3)^2$.

We now present the resources for partial product creation, the first half of a modular multiplier, including the reverse computation.

The circuit depth is $O(\log n)$:

$$D_{PPC} = 32 \log_2 n + 150. \tag{1.11}$$

The module depth is $O(1)$:

$$\overline{D}_{PPC} = 8. \tag{1.12}$$

The circuit size is $O(n \log^2 n)$:

$$\begin{aligned} S_{PPC} &= (6n+9)\log_2 n + & (1.13)\\ &\quad (26n^3 + 232n^2 + 224n + 159). & (1.14) \end{aligned}$$

The module size is $O(n^2)$:

$$\overline{S}_{PPC} = 6n^2 + 26n + 19. \tag{1.15}$$

The circuit width is $O(n^3)$:

$$W_{PPC} = 6n^3 + 48n^2 - 8n + 1. \tag{1.16}$$

The module width is $O(n^2)$:

$$\overline{W}_{PPC} = 2n^2 + 14n + 9. \tag{1.17}$$

1.6.3 Modular Multiple Addition

As a subroutine to modular multiplication, we define the operation of repeatedly adding multiple numbers down to a single CSE number, called *modular multiple addition*.

The modular multiple addition circuit generically adds down $t' \times n$-bit conventional numbers to an n-bit CSE number:

$$z^{(1)} + z^{(2)} + \ldots z^{(t')} \equiv (u + v)[m]. \tag{1.18}$$

It does not matter how the t' numbers are generated, as long as they are divided into groups of three and have their bits interleaved to be the inputs of a CSA tile. From the previous section, serial multiplication results in $t' \leq n$ and parallel multiplication results in $t' \leq n^2$. Each CSA tile is contained in its own module. These modules are arranged in layers within a logarithmic depth binary tree, where the first layer contains $\lceil t'/3 \rceil$ modules. A modular addition occurs in all the modules of the first layer in parallel. The outputs from this first layer are then teleported to be the inputs of the next layer of modules, which have at most two-thirds as many modules. This continues until the tree terminates in a single module, whose output is a CSE number $u + v$ which represents the modular product of all the original t' numbers. The resulting height of the tree is $(\lceil \log_{3/2}(t'/3) \rceil + 1)$ modules.

As the parallel modular additions proceed by layers, all previous layers must be maintained in a coherent state, since the modular addition leaves garbage bits behind. Only at the end of modular multiple addition, after the final answer $u + v$ is obtained, can all the previous layers be uncomputed in reverse to free up their ancillae.

These steps are best illustrated with a concrete example in Figure 1.8. The module for each CSA tile is represented by the symbol from Figure 1.7. The arrows indicate the teleportation of output numbers from the source tile to be input numbers into a destination tile.

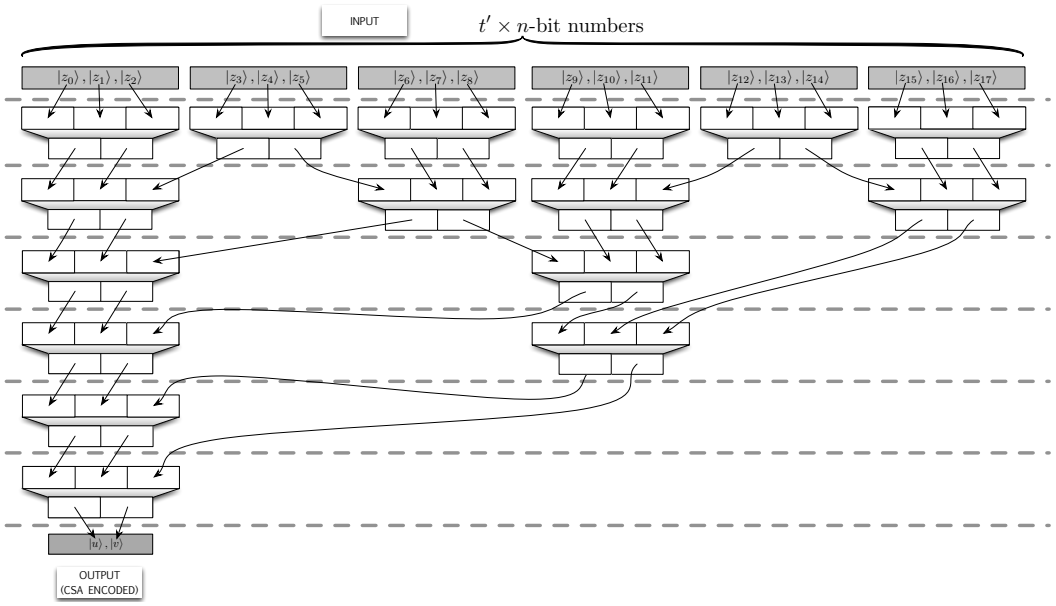

Figure 1.8: Modular multiple addition of quantum integers on a CSA tile architecture for $t' = 18$ in a logarithmic-depth tree with height $(\lceil \log_{\frac{3}{2}}(t'/3) \rceil + 1) = 6$. Arrows represent teleportation in between modules.

Now we can analyze the circuit resources for multiplying n-bit quantum integers, which requires $(t' - 2)$ modular additions, for t' from Equation 1.10. The circuit width is the sum of the $O(n^3)$ ancillae needed for partial product creation and the ancillae required for $O(n^2)$ modular additions. Each modular addition has width $O(n)$ and depth $O(1)$ from the previous section. There are $\lceil \log_{3/2}(n^2/3) \rceil + 1$ timesteps of modular addition. Therefore the entire modular multiplier circuit has depth $O(\log n)$ and width $O(n^3)$.

1.6.4 Modular Multiplier Resources

.

The circuit depth of the entire modular multiplier is $O(\log n)$:

$$D_{MM} = 1383 \log_2 n + 3930. \tag{1.19}$$

The module depth is $O(\log n)$:

$$\overline{D}_{MM} = 2\log_2 n + 11. \tag{1.20}$$

The circuit size is $O(n^3)$:

$$S_{MM} = (6n+9)\log_2 n + \tag{1.21}$$

$$(1152n^3 + 10780n^2 + 17628n + 7082). \tag{1.22}$$

The module size is $O(n^3)$:

$$\overline{S}_{MM} = 15n^3 + 127n^2 + 178n + 50. \tag{1.23}$$

The circuit width is $O(n^3)$:

$$W_{MM} = 66n^3 + 558n^2 + 870n + 290. \tag{1.24}$$

The module width is $O(n^2)$:

$$\overline{W}_{MM} = 4n^2 + 28n + 15. \tag{1.25}$$

1.7 Quantum Modular Exponentiation

We now extend our arithmetic to modular exponentiation, which is repeated modular multiplication controlled on qubits supplied by a phase estimation procedure. If we wish to multiply an n-qubit quantum input number $|x\rangle$ by t classical numbers $a^{(j)}$, we can multiply them in series. This requires depth $O(t \log n)$ in modular multiplication operations.

Now consider the same procedure, but this time each classical number $a^{(j)}$ is controlled on a quantum bit p_j. This is a special case of multiplying by t quantum integers in series, since a classical number entangled with a quantum integer is also quantum. It takes the same depth $O(t \log n)$ as the previous case.

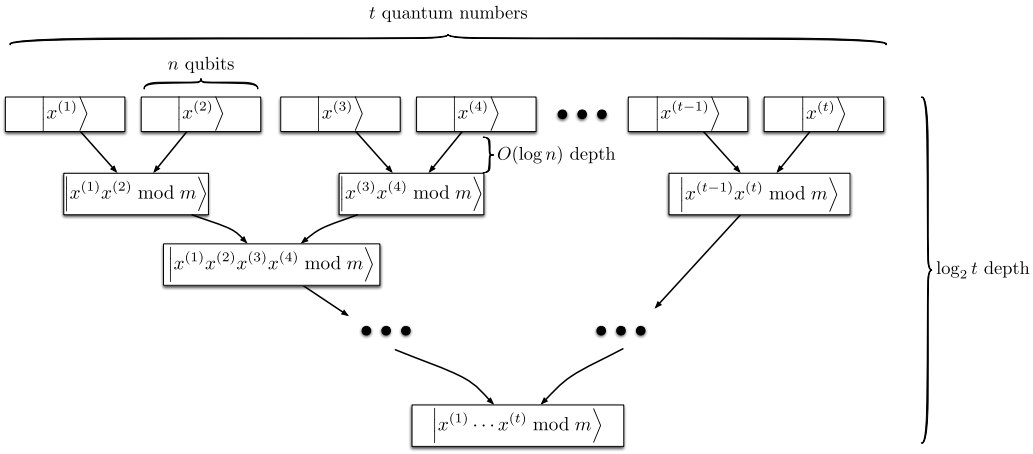

Figure 1.9: Parallel modular exponentiation: multiplying t quantum integers in a $O(\log(t)\log(n))$-depth binary tree. Arrows indicate modular multiplication.

Finally, we consider multiplying t quantum integers $\{x^{(1)}, x^{(2)}, \ldots, x^{(t-1)}, x^{(t)}\}$ in a parallel, logarithmic-depth binary tree. This is shown in Figure 1.9, where arrows indicate multiplication. The tree has depth $\log_2(t)$ in modular multiplier operations. Furthermore, each modular multiplier has depth $O(\log(n))$ and width $O(n^3)$ for n-qubit numbers. Therefore, the overall depth of this parallel modular exponentiation structure is $O(\log(t)\log(n))$ with width $O(tn^3)$. In phase estimation for QPF, it is sufficient to take $t = O(n)$ [76, 53]. Therefore our total depth is $O(\log^2(n))$ and our total size and total width are $O(n^4)$, as desired. At this point, combined with the parallel phase estimation procedure of [53], we have a complete factoring implementation in our 2D nearest-neighbor architecture in polylogarithmic depth.

We will now calculate numerical constants to upper bound circuit resources.

According to the Kitaev-Shen-Vyalyi parallelized phase estimation procedure [53], for a constant success probability of 3/4, it is sufficient to multiply together $t' = 2867n$ quantum integers, controlled on the qubits $|p_j\rangle$, in parallel.

In Section 1.7.1, we describe the last step of modular exponentiation in CSE. In Section 1.7.2, we state the final circuit resources for the entire modular exponentiation circuit, and therefore, our quantum period-finding procedure.

1.7.1 Converting Back to a Unique Conventional Number

The final product of all t quantum integers is in CSE which is not unique. As stated in Gossett's original paper [41], this must be converted back to a conventional number using, for example, the quantum carry-lookahead adder (QCLA) from [31]. We can convert this to a nearest-neighbor architecture by using the qubit reordering construction of [82]. We now compute the resources needed for this last step.

To add two $(n+2)$-bit numbers in a QCLA, we have a circuit width of $k = (4(n+2) - 2\log_2 n - 1)$. The depth is at most $4\log_2 n + 2$ gates, and some of them act on qubits that are not nearest-neighbors. Therefore, we add in between each gate a reordering circuit that takes k^2 (reusable) ancillae qubits and uses two rounds of constant-depth teleportation to rearrange the qubits into a new order where all the gates are nearest-neighbor. Adding in the teleportation circuit resources from Table 1, we can calculate the following resources.

The circuit depth is $O(\log n)$:

$$56\log_2 n + 28. \tag{1.26}$$

The circuit size is $O(n^2 \log n)$:

$$\begin{aligned} 96\log_2^3 n \quad &- \quad (384n + 624)\log_2^2 n \\ &+ \quad (384n^2 + 1152n + 840)\log_2 n \\ &+ \quad (192n^2 + 672n + 588). \end{aligned} \tag{1.27}$$

The circuit width is $O(n^2)$:

$$4\log_2^2 n - (16n + 30)\log_2 n + 16n^2 + 60n + 56. \tag{1.28}$$

1.7.2 *Circuit Resources for Modular Exponentiator*

This leads to the following circuit resource upper bounds for a modular exponentiator. Therefore, these are the total resources for running a single round of parallel QPF as part of Shor's factoring algorithm.

The circuit depth is $O(\log^2 n)$:

$$D_{ME} = 1383 \log_2^2(n) + 21253 \log_2(n) + 49095. \tag{1.29}$$

The module depth is $O(\log n)$:

$$\overline{D}_{ME} = 3 \log_2 n + 24 \tag{1.30}$$

The circuit size is $O(n^4)$:

$$\begin{aligned} S_{ME} &= 96 \log_2^3 n + \\ &- (384n + 624) \log_2^2 n \\ &+ (384n^2 + 1152n + 840) \log_2 n \\ &\quad 3302324n^4 + 30900797n^3 + 50521837n^2 + 20284306n + -6494. \end{aligned} \tag{1.31}$$

The module size is $O(n^2)$:

$$\overline{S}_{ME} = 5749n^2 + 8725n + 175. \tag{1.32}$$

The circuit width is $O(n^4)$:

$$W_{ME} = 94598n^4 + 799749n^3 + 1246692n^2 + 415222n - 145. \tag{1.33}$$

The module width is $O(n)$:

$$\overline{W}_{ME} = 1434n. \tag{1.34}$$

1.8 Asymptotic Results

The asymptotic resources required for our approach, as well as the resources for other nearest-neighbor approaches, are listed in Table 1.2, where we assume a fixed constant error probability for each round of QPF. Not all resources are provided directly by the referenced source.

Resources in square brackets are inferred using Equation 27. These upper bounds are correct, but may not be tight with the upper bounds calculated by their respective authors. In particular, a more detailed analysis could give a better upper bound for circuit size than the depth-width product. Also note that the work by Beckman et al. [9] is unique in that it uses efficient multi-qubit gates inherent to linear ion trap technology which at first seem to be more powerful than 1D NTC. However, use of these gates does not result in an asymptotic improvement over 1D NTC.

We achieve an exponential improvement in nearest-neighbor circuit depth (from quadratic to polylogarithmic) with our approach at the cost of a polynomial increase in circuit size and width. Similar depth improvements at the cost of width increases can be achieved using the modular multipliers of other factoring implementations by arranging them in a parallel modular exponentiator. Our approach is the first implementation for factoring on 2D NTC, augmented with a classical controller and parallel, communicating modules (2D CCNTCM).

1.9 Conclusion

Insert acknowledgements for 2D factoring work, mention paper in which these results appear.

Implementation	Architecture	Depth	Size	Width
Vedral, et al. [103]	AC	$[O(n^3)]$	$O(n^3)$	$O(n)$
Gossett [41]	AC	$O(n \log n)$	$[O(n^3 \log n)]$	$O(n^2)$
Beauregard [8]	AC	$O(n^3)$	$O(n^3 \log n)$	$O(n)$
Zalka [108]	AC	$O(n^2)$	$[O(n^3)]$	$O(n)$
Takahashi & Kunihiro [96]	AC	$O(n^3)$	$O(n^3 \log n)$	$O(n)$
Cleve & Watrous [26]	AC	$O(\log^3 n)$	$O(n^3)$	$[O(n^3 / \log^3 n)]$
Beckman et al. [9]	Ion trap	$O(n^3)$	$O(n^3)$	$O(n)$
Fowler, et al. [37]	1D NTC	$O(n^3)$	$O(n^4)$	$O(n)$
Van Meter & Itoh [99]	1D NTC	$O(n^2 \log n)$	$[O(n^4 \log n)]$	$O(n^2)$
Kutin [59]	1D NTC	$O(n^2)$	$O(n^3)$	$O(n)$
Current Work	2D CCNTCM	$O(\log^2 n)$	$O(n^4)$	$O(n^4)$

Table 1.2: Asymptotic circuit resource usage for quantum factoring of an n-bit number.

Chapter 2

NEAREST-NEIGHBOR FACTORING IN SUBLOGARITHMIC DEPTH

It is now natural to ask: given such dramatic improvement in circuit depth for nearest-neighbor factoring from quadratic [59] to polylogarithmic in the last chapter, can we decrease depth further? Surprisingly, the answer is yes. In this chapter, we now decrease the depth below polylogarithmic, in fact, to be $O((\log\log n)^2)$. To do this, we take inspiration from two main lines of related work. First, it known how to compute many useful arithmetic functions, including those used in modular exponentiation, in constant depth by introducing a threshold gate. Second, a similar construction (on AC) gives us a quantum OR gate. Using these results, we construct a *quantum majority gate* on 2D CCNTCM to achieve quantum modular exponentiation, and therefore factoring, in the above depth.

In Section 2.1 we provide background for classical circuit complexity, including common universal gate sets, how they allow us to define circuit complexity classes, and the relationships between those classes. We also discuss the powerful threshold gate and its variations. Finally, we provide quantum analogues for all these notions. The quantum threshold gate can be decomposed into simpler operations, namely CNOT and arbitrary single-qubit gates, while maintaining constant-depth.

However, even this simple gate set must be compiled down to 2D CCNTCM. This accounts for the discrepancy between the non-constant quantum depth upper bound and the constant classical depth lower bound mentioned above. Our solution is to augment our factoring circuit with quantum compiler modules using the Kitaev-Shen-Vyalyi method [53] and the programmable ancillae rotation method of Jones et al. [51]. We discuss this special case of quantum compiling overhead and its effect on our factoring implementation in Section 2.2.

We then discuss our main result in Section 2.3, a 2D CCNTCM implementation of a quantum majority gate with fanin n having circuit depth $O((\log\log n)^2)$ and circuit

Figure 2.1: An example of a classical circuit implementing a Boolean function.

size and width $O(n^2 \log^2 n)$. Using this quantum majority gate and the constant-depth majority circuits of Reif-Tate [81] and Yeh-Varvarigos [107], we achieve a complete circuit for quantum modular exponentiation. We conclude in Section 2.4 with open problems related to factoring architectures.

2.1 Circuit Complexity Background

A circuit can be thought of as a directed acyclic graph in which the nodes are logical gates drawn from a certain (universal) set and the edges represent the connection of the output of one gate to the input of another gate. This graph is not equivalent to, but is related to, the graph of an architecture as described in Chapter 1. The edges of a circuit graph can be mapped to the nodes (qubits) of an architectural graph; the nodes of the circuit graph, for 2D CCNTC, can be mapped to single nodes or connected pairs of nodes in the architectural graph. The notion of a gate and a circuit are not mutually exclusive: both implement a boolean function, but a gate is usually treated as a primitive, subcircuit element for the purpose of counting resources in a larger circuit.

We can also define special nodes which are not gates, but rather are placeholder "sources" which provide the inputs to the circuit (they only have out-degree) and "sinks" which consume the outputs to the circuit (they only have in-degree). The in-degree of a node is also known as its *fanin* and the out-degree of a node is also known as its *fanout*. Classical circuits implement boolean functions, which take in n input bits to one output bit.

$$f : \{0,1\}^n \to \{0,1\} \tag{2.1}$$

We denote a gate by its fanin as a subscript and other optional parameters as super-

scripts (n and k, respectively, in the equation below).

$$\mathrm{GATE}_n^k \tag{2.2}$$

The fanin n will be neglected where it is obvious, such as for the following well-known gate set which is universal for classical circuits:

- $\mathrm{NOT} = \mathrm{NOT}_1$
- $\mathrm{AND} = \mathrm{AND}_2$
- $\mathrm{OR} = \mathrm{OR}_2$

2.1.1 Gates Based on the Hamming Weight

While the gates above have a simple truth table, it is useful to describe a wider class of gates with general fanin n in a compact way as a function on the input Hamming weight. These include the gates named below, which are 1 when the condition to their right is met, and 0 otherwise.

- The logical OR gate $\mathrm{OR}_n : |x| > 0$
- The logical AND gate $\mathrm{AND}_n : |x| = n$
- The modular gate $\mathrm{MOD}[q]_n : |x| \bmod q = 0$
- The parity gate $\mathrm{PA}_n : |x| \bmod 2 = 0$
- The exact gate $\mathrm{EX}_n^t : |x| = t$
- The threshold gate $\mathrm{TH}_n^t : |x| \geq t$
- The majority $\mathrm{MAJ}_n : |x| \geq n/2$

Many of these gates are also related in interesting ways, as noted in [97]. OR_n and EX_n^0 are negations of each other. PA_n is equivalent to $\mathrm{MOD}[2]_n$. TH_n^t can be implemented with $n-t$ parallel copies of EX_n^k for $t \geq k \geq n$ followed by PA_{n-t} on the outputs.

2.1.2 *Classical Circuit Complexity Classes*

We have introduced a menagerie of interesting gates, all of which are universal with the *NOT* gate. However depending on what gates are in our universal set, our circuits may have different depth and size. Therefore, we define complexity classes of circuits based on the allowed gate set and study more general relationships among these classes. As is usual in the literature, we will interchangeably use "circuits" to mean uniform circuit families parameterized by their input size n. This will let us formalize the notion of which gates are more powerful than other gates and which are equivalent.

In classical circuits, we take unbounded fanout for granted (any node can have arbitrary out-degree). These are common in the literature of classical circuits. We will list them in order of the size of their universal set, where each subsequent class adds more gates.

NC: circuits consisting of NOT_1 and AND_2 and OR_2 gates.

AC: NC circuits augmented with AND_n and OR_n gates, for $n \geq 2$.

AC[q]: AC circuits augmented with $\mathrm{MOD}[q]_n$ gates.

ACC: the union of $\mathsf{AC}[q]$ for all positive integers $q > 2$.

TC: AC circuits augmented with TH_n^t gates, for $n \geq 2$ and $0 \leq t \leq n$.

We are often interested in the computing power of the above circuit classes restricted in some way, usually shallow depth. We denote by a superscript k a complexity class of functions implementable by circuits of depth bounded by $O(\log^k n)$.

For these classical circuit classes, it is known that containment is proper between NC^0, AC^0, and TC.

$$\mathsf{NC}^0 \subsetneq \mathsf{AC}^0 \subsetneq \mathsf{TC}^0 \tag{2.3}$$

For example, the function AND_n and OR_n is known *not* to be in NC^0 but is in AC^0 by definition. Likewise, the gate PA_n is known *not* to be in AC^0 but is in TC^0 [23].

It is also known that $\mathsf{AC}^0[q] \neq \mathsf{AC}^0[p]$ for p and q being powers of distinct primes [91].

2.1.3 Linear Threshold Elements

For historical reasons, we will mention here a more general version of TH_n^t which was studied extensively in the 1990s. This threshold gate, called a *linear threshold element* (LTE), is a boolean function defined as the sign of a weighted sum of its input bits. There is a weight not associated with any input, called a bias.

$$f : 0,1^n \to 0,1 = \operatorname{sgn}\left(w_0 + \sum_{i=1}^{n} w_i x_i\right) \tag{2.4}$$

The LTE was biologically inspired by neurons in the human brain and subsequently the neural network model of computation. Computational neurons are simple elements which can be combined in highly parallel, shallow depth (less than 6) networks to compute seemingly complicated arithmetic functions such as (multiple) addition, (multiple) multiplication, division, and comparison. These results are summarized in Siu et al. [88] based on techniques by Beame et al. [7]. LTE's are arguably the predecessors of logic elements (LE's) in modern day field-programmable gate-arrays (FPGA's).

However, the most general kind of LTE can have real-valued weights, which can be difficult to implement fault-tolerantly on digital logic. A single LTE's power can be changed dramatically by restricting its weights, but this power goes away when we consider *threshold circuits* of LTEs restricted to constant depth. Through a succession of results, it was shown that restricting the weights to be rational numbers bounded by a polynomial in n [89] did not decrease the power of such constant-depth threshold circuits. That is, any

threshold circuit with unrestricted-weight LTE's could be simulated with a constant-depth overhead with restricted-weight LTE's.

Remarkably, this constant-depth simulability applies even when restricting LTE's to unit weight (weights of only $+1$) and integer bias $0 \leq t \leq n$, which is equivalent to the definition of TH_n given above. It even applies when we restrict the threshold to be $\lceil n/2 \rceil$, which is the definition of MAJ_n given above [40]. In fact, where many arithmetic functions above have been shown to be implementable in majority circuits [81, 107]. Therefore, without loss of generality, we can concentrate on majority circuits for the remainder of this paper.

2.1.4 Quantum Gates

We can define analogous quantum circuit complexity classes by considering circuits over (reversible) quantum gates. From now on, when we refer to a gate GATE_n^k, we mean its quantum version which behaves as follows on an n-qubit "input" register $|x\rangle$ and a single-qubit "output" register $|y\rangle$. We denote the single-bit output of the classical function $\mathrm{GATE}_n^k(x)$ as $g(x)$.

$$\mathrm{GATE}_n^k |x\rangle |y\rangle \rightarrow |x\rangle |y \oplus g(x)\rangle \tag{2.5}$$

Unbounded fanout is taken for granted in classical circuit classes because it is physically realistic to implement using electrical devices. However, for quantum circuits, we must make use of unbounded quantum fanout (described in detail in Chapter 1) to create entangled copies of the output qubit for every subsequent gate which consumes it as an input. We can even define the gate FANOUT_n on the source qubit x onto target qubits $y_1, \ldots, y_n$.

$$\mathrm{FANOUT}_n |x\rangle |y_1\rangle \ldots |y_n\rangle \rightarrow |x\rangle |y_1 \oplus x\rangle \ldots |y_n \oplus x\rangle \tag{2.6}$$

There is not a clear, unique quantum version of the classical circuit classes above. However, the following quantum circuit complexity classes, taken from [47] begin with

the simplest universal set and augment it with quantum versions of the corresponding classical gates. Each class has a subscript f to indicate the inclusion of FANOUT$_n$.

QNC^0_f: constant-depth quantum circuits consisting of CNOT and single-qubit gates.

QAC^0_f: constant-depth QNC^0_f circuits augmented with quantum AND$_n$ and OR$_n$ gates, for $n \geq 2$.

QAC[q]^0_f: constant-depth QAC^0_f circuits augmented with quantum MOD$[q]_n$ gates, for $n \geq 2$.

QACC^0_f: the union of all QAC[q]^0_f for positive integers q.

QTC^0_f: constant-depth QAC^0_f circuits augmented with TH^t_n gates, for $n \geq 2$ and $0 \leq t \leq n$.

It is an important result of Takahashi-Tani [97] that the presence of FANOUT$_n$ equalizes the three classes below.

$$\mathsf{QNC}^0_f \subsetneq \mathsf{QAC}^0_f \subsetneq \mathsf{QTC}^0_f \tag{2.7}$$

Moore shows that parity is equivalent to fanout in a quantum setting [73].

$$\mathsf{QAC}^0_f = \mathsf{QAC}[2]^0_f \tag{2.8}$$

Furthermore, it is has been discovered by Green et al. that including any MOD$[q]^0_f$ leads to the same complexity class in constant depth.

$$\mathsf{QACC}^0_f = \mathsf{QAC}[q]^0_f \forall q \geq 2 \tag{2.9}$$

2.2 *Controlled Rotations for Factoring*

In Section 1.3 of Chapter 1, we stated that we wished to avoid factoring implementations that used a QFT due to the fine single-qubit rotations involved. Due to the requirements of fault-tolerance on a particular physical implementation, we can usually implement only

a set of gates that is fixed (it does not change with the problem input size), discrete (of finite size), and universal. This last property is necessary for us to approximate any other gate *not* in our set, especially single-qubit phase rotations of the form $\Lambda(e^{i\phi})$. Such an approximation would involve a quantum compiling procedure, such as Solovay-Kitaev, which is the subject of Chapter 3. However, we mention it here because the choice of our universal set determines the true depth of any circuit.

2.2.1 *Controlled-R_Z Rotations*

In our polylogarithmic factoring implementation, we were able to reduce all our arithmetic circuits to such a fixed, discrete universal set. These arithmetic circuits are discrete and classical and nature, so it is not surprising that we can implement them in a discrete way. However, to reduce the depth further, we need to introduce the idea of a quantum threshold gate, which requires controlled-rotations similar to the QFT. This controlled-rotation gate is shown in Figure 2.3 along with its decomposition into CNOTs and single-qubit $R_Z(\phi)$ rotations. These rotations depend on the input size.

If we were given pre-generated ancillae of the form $|0\rangle + e^{i\phi}|1\rangle$, we could use the method of programmable ancillae rotation (PAR) from [51] to apply the gate $R_Z(\phi)$ probabilistically using the circuit from Figure 2.2. Note that we have a 50% chance of applying the opposite rotation $-\phi$, and so we must chain several rounds of the given circuit. In expectation, we achieve the desired rotation in a constant depth of 2, and now we must consider all circuit resources in terms of expected (average) values. The probability of k PAR rounds all failing (and the need to run a more expensive KSV quantum compiling procedure) is 2^{-k}, giving the following expected depth. For $k = O(1)$, one can make the expected depth of both PAR and quantum compiling grow with an arbitrarily small multiplicative constant. However the asymptotic cost will still track that of whatever quantum compiling procedure is used. For $k = \omega(1)$, we make the expected depth of quantum compiling constant; unfortunately, now the expected depth of PAR rounds is no longer constant. This is an inherent tradeoff in using PAR. An analogous calculation can be done

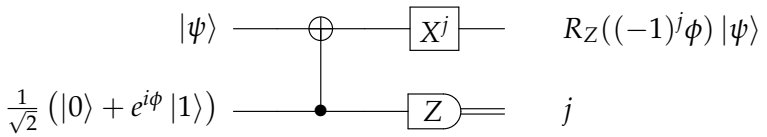

Figure 2.2: One round of programmable ancillae rotation (PAR) to probabilistically achieve arbitrary single-qubit rotations [51]

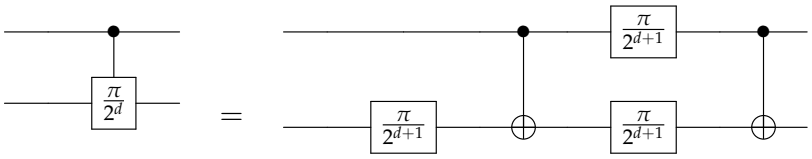

Figure 2.3: Decomposition of a controlled-R_z rotation

for the size.

$$\left(\sum_{m=1}^{k} \frac{1}{2^m} O(1)\right) + 2^{-k} O((\log\log n)^2) \tag{2.10}$$

2.2.2 *Quantum Compiler Module*

We can augment any 2D CCNTCM circuit $\mathcal{C}$ with a set of modules that run a separate quantum compiler procedure. This quantum compiler is a source of ancillary qubits of the form $|0\rangle + e^{i\phi}\,|1\rangle$. The angles ϕ are determined by the original $\mathcal{C}$ and are known in advance (classically-precomputed, or non-adaptive). In the case of majority gates for factoring, as described in Section 2.3, the angles are of the form $\phi_k = \frac{2\pi}{2^k}$ where $k \leq (\log_2 n' + 2)$ for a majority gate with fanin n'. Since this represents the most fine-grained resolution of rotation, our allowable error is $2^{-(k+1)} = 2^{-(\log_2 n' + 3)}$.

This quantum compiler resource has a parameter ϵ which is the allowable error in the angles of the return PAR qubits. We assume they have some precision $\epsilon = 2^{-k}$ from the

Circuit Size	$O((\log n')^2 \log\log n')$
Circuit Depth	$O((\log\log n')^2)$

Table 2.1: KSV quantum compiling resources for single-qubit rotations in majority gates of fanin n'.

true desired angle $\tilde{\phi}$.

$$|\phi - \tilde{\phi}| \leq \epsilon = 2^{-(k+1)} = 2^{-(\log_2 n' + 3)} \tag{2.11}$$

.

Often, quantum compiler resources are given in terms of the quantity $(1/\epsilon)$, which we can re-express in terms of the majority gate fanin as $O(1/n')$. While we will delay a discussion and comparison of quantum compilers until Chapter 3, for now we assume an upper bound of the Kitaev-Shen-Vyalyi (KSV) algorithm using parallel phase estimation from Section 13 of [53]. This has the resources given in Table 2.1.

As we will discover later, the fanin of each majority gate is linear in the size of the modulus to factor: $n' = O(n)$. Therefore, if we are not allowed PAR qubits for free in our model, our construction will actually have minimum depth $O((\log\log n)^2)$, which is not constant, but is still sublogarithmic. This relationship still holds if the fanin is merely $n' = poly(n)$. It is still an open question [47] whether a constant-depth quantum majority gate exists even on AC with a fixed finite basis, let alone NTC architectures and their sub-models.

From the argument in the previous section: even though the PAR procedure is probabilistic, the asymptotic circuit resources required can be deterministically upper-bounded.

2.3 *Quantum Majority Circuits for Modular Exponentiation*

A quantum majority circuit is made from quantum MAJ_n gates. As mentioned before, depth-k majority circuits are equivalent in power to depth-k LTE circuits with polynomially-

bounded weights: $\mathsf{MAJ}_k = \hat{\mathsf{LT}}_k$ [3, 40].

We are also interest in majority gates of polynomial fanin. In a majority circuit, the fanin of any one majority gate is bounded above by the circuit size, since in the worst case, one gate receives an output from every other gate as input. As long as we restrict ourselves to polynomial-size circuits, we are assured that our circuit fanin is also polynomial. This is the primary consideration in achieving sublogarithmic depth for quantum compiling single-qubit rotations, and therefore for factoring.

We now contribute a quantum majority circuit for modular exponentiation of an n-bit modulus on 2D CCNTCM. In this section, we show that each majority circuit can be implemented in a single module with $O(n)$ qubits. Any reordering needed between the output qubits of one timestep in a majority circuit and the input qubits of the next timestep is handled by teleportation in between modules, as allowed on 2D CCNTCM. Therefore, it suffices for us to do the following:

1. Show that a (classical) majority circuit exists for (classical) modular exponentiation. We can translate this to a 2D CCNTCM architecture where each MAJ_n gate is translated to $O(1)$ modules of $\Omega(n)$ width.

2. Show a 2D CCNTCM quantum architecture for a single MAJ_n gate in $O(1)$ modules of $\Omega(n)$ width, assuming the incoming teleportation of qubits of this form $\frac{1}{\sqrt{2}}(|0\rangle + e^{i\phi}|1\rangle)$. These qubits are used to perform the corresponding single-qubit rotations using the procedure of Jones et al. [51].

3. Include the circuit resources for (KSV) quantum compiler modules that can produce the qubits in the previous step.

We do the first item by relying on the following two theorems from Yeh-Varvarigos, which we re-state below without proof. Both of these theorems allow an extra parameter $\epsilon \in (0, 1]$ which determines the tradeoff between circuit depth and circuit size. They apply to *classical majority circuits*. Therefore, we must scale all their circuit resources by those for a quantum majority gate, calculated in Section 2.3.4.

Theorem 1. ***(Yeh-Varvarigos) Multiple product in constant depth and polynomial size: [107].*** *The n^2-bit product of $n \times n$-bit numbers can be computed by a majority circuit of depth $O(\frac{1}{\epsilon})$, size and width $O(\frac{1}{\epsilon}n^{3+2\epsilon})$, and fanin $O(n)$.*

Theorem 2. ***(Yeh-Varvarigos) Modular reduction in constant depth and polynomial size [107].*** *The n-bit binary representation of the modular residue $x \bmod m$, where x is an n^2-bit number and m is an n-bit modulus, can be computed by a majority circuit of depth $O(\frac{1}{\epsilon})$, size and width $O(\frac{1}{\epsilon}n^{1+2\epsilon})$, and fanin $O(n^2)$.*

Both of these theorems rely on a Chinese Remainder representation for an n-bit number. A conventional binary representation of a number treats bits as coefficients for weights of $O(2^n)$, which requires exponential weight to represent in an LTE. In a Chinese Remainder representation, a number is given a "mixed-radix" representation, where each coefficient is associated with a modular residue for a prime with $O(n)$ bits. In this way, the weights needed to represent each coefficient are bounded polynomially and not exponentially. A more detailed reference of this technique can be found in [81].

We delay discussion of quantum compiler considerations until Section 2.2.

We now accomplish the second item (a concrete architecture for a quantum majority gate) in the remainder of this section by a sequence of building blocks, each on 2D CCNTC with constant depths and polynomially-bounded sizes and widths.

- a $\mathrm{BIAS}_n^{t,\phi}$ gate which distinguishes between $|x| = t$ and $|x| = (\lceil n/2 \rceil - t) \bmod n$ with a measurement bias of $e^{i\phi}$. This is described in Section 2.3.1.

- an $\mathrm{EX}^t_{n \to \log_n}$ gate which reduces from EX^t_n (on n qubits) to $\mathrm{EX}^t_{\log n}$ (on $\lceil \log_2(n+1) \rceil$ qubits). This is described in Section 2.3.2.

- an $\mathrm{EX}^t_{\log_n}$ gate which acts on $O(\log n)$ qubits. This is described in Section 2.3.3.

2.3.1 BIAS Gate

We define the BIAS gate using the results in [47], where it is called a $\mu_\phi^{|x|-t}$ gate. We can also think of it as rotating the output qubit $|+\rangle$ by Hamming weight with a threshold t

Figure 2.4: The layout for a BIAS gate on 2D CCNTC.

subtracted. It operates as follows:

$$\mathrm{BIAS}_n^{t,\phi} |x\rangle |+\rangle \to |x\rangle \left|\mu_\phi^{|x|-t}\right\rangle \tag{2.12}$$

The output qubit begins in the state $|+\rangle$, which has equal probability of being measured in the $|0\rangle$ state or the $|1\rangle$ state. It ends in the following state, which introduces a bias between measuring $|0\rangle$ or $|1\rangle$ proportional to the difference $(|x| - t)$.

$$\left|\mu_\phi^{|x|-t}\right\rangle = \frac{1 + e^{i\phi(|x|-t)}}{2} |0\rangle + \frac{1 - e^{i\phi(|x|-t)}}{2} |1\rangle \tag{2.13}$$

When $\phi = 2\pi/n$, then the BIAS gate allows us to distinguish between the case of $|x| = t$ or $|x| = (\lceil n/2 \rceil - t) \bmod n$. As we will see in the next section, rotations by multiples of $2\pi/n$ will allow us to reduce the size of an EX_n^t gate.

Theorem 3. ***Constant-depth BIAS gate.*** *The $BIAS_n^{t,\phi}$ gate can be implemented on 2D CCNTCM with a depth of $O(1)$, a size and width of $O(n)$, and expected $O(n)$ teleported PAR qubits of the form $(|0\rangle + e^{i\phi} |1\rangle)$ and $(|0\rangle + e^{-i\phi \cdot t} |1\rangle)$.*

Proof. We can lay out the circuit from Figure 1 in [97] on a 2D CCNTC lattice as shown in Figure 2.4. The size includes a Hadamard to transform the output qubit into $|+\rangle$, a fanout of this qubit over $n + 1$ qubits, $O(n)$ gates to apply the rotations $R_Z(\phi)$ and $O(1)$ gates to apply the rotation $R_Z(-\phi \cdot t)$, and a corresponding unfanout, which is $O(n)$ total. This occurs on $O(n)$ qubits, and can be arranged to take $O(1)$ depth. This circuit is contained within a single module. □

2.3.2 EX Logarithmic Reduction

Now we wish to show how to reduce EX_n^t gate (on n qubits) to an $\mathrm{EX}_{\log n}^t$ gate (on $O(\log n)$ qubits). That is, we wish to implement the following gate $\mathrm{EX}_{n \to \log_2 n}^t$ on an n-qubit input register $|x\rangle$ to produce an m-qubit output register $|y\rangle$, where $m = \lceil \log_2 n + 1 \rceil$. Running

EX_n^t on $|x\rangle$ should produce the same output qubit $|z\rangle$ as running EX_m^t on $|y\rangle$. This is formally defined below.

$$\mathrm{EX}_{n\to\log_2 n}^t |x\rangle |0^m\rangle \quad \to \quad |x\rangle |y\rangle \tag{2.14}$$

$$\mathrm{EX}_n^t |x\rangle |0\rangle \quad \to \quad |x\rangle |z\rangle \tag{2.15}$$

$$\mathrm{EX}_m^t |y\rangle |0\rangle \quad \to \quad |y\rangle |z\rangle \tag{2.16}$$

Theorem 4. ***Constant-depth EX reduction gate.*** *The $EX_{n\to\log_2 n}^t$ gate can be implemented on 2D CCNTCM with a depth of $O(1)$, a size and width of $O(n \log n)$, and expected $O(n \log n)$ teleported PAR qubits of the form $(|0\rangle + e^{i\phi_k} |1\rangle)$ and $(|0\rangle + e^{-i\phi_k \cdot t} |1\rangle)$*

Proof. We map the construction from Theorem 19 in [47] onto 2D CCNTCM. This step involves $m = \lceil \log_2(n+1) \rceil$ parallel $\mathrm{BIAS}_n^{t,\phi_k}$ gates from the last section for $1 \leq k \leq m$, where $\phi_k = \frac{2\pi}{m} k$, each in their own modules. This maintains constant circuit depth while only increasing circuit size by a $O(\log n)$ factor. □

2.3.3 OR on Logarithmic Qubits

We now map an exact OR gate from Takahashi-Tani [97] to 2D CCNTCM. Unlike the approximate OR gate (and its variation, an approximate EXACT gate) from Hoyer-Spalek [47], it is both complete and sound with probability 1 and it completes in constant depth. However, it has size exponential in its input size n': $O(n' 2^{n'})$. Therefore, the previous logarithmic reduction in Section 2.3.2 is necessary to reduce $O(n)$ qubits, the fanin of our majority circuit, to $n' = O(\log n)$, to give us a polynomial circuit size of $O(n \log n)$. We will denote this gate as $OR_{\log n}^t = OR_{n'}^t$ to emphasize that it can only be efficiently used when the number of qubits have been reduced to be logarithmic in the input size of the overall problem (in this case, factoring). The circuit depth will be constant no matter what, but circuit size is only polynomially-bounded if the previous condition is met.

Theorem 5. ***Constant-depth exact OR gate on logarithmic qubits.*** *The $OR_{\log n}^t$ gate can be implemented on 2D CCNTCM with a circuit depth of $O(1)$, a circuit size and width of $O(n \log n)$,*

and expected $O(n \log n)$ teleported PAR qubits of the form $(|0\rangle + e^{i\frac{2\pi}{O(n)}} |1\rangle)$. This takes module depth of $O(1)$, module size of $O(n \log n)$, and module width $O(n \log n)$.

Proof. Following the notation of Lemma 2 from [97], we have two modules:

1. One module contains the input qubits x_i and the R_j and S registers. We will call this the RS module.

2. One module contains the T register, which we call the T module.

First, we consider the RS module. This has a total of $n'2^{n'} = O(n \log n)$ qubits each. We use a sorting network from Rosenbaum [82] to reorder qubits and allow for nearest-neighbor interactions. This requires size and width $O(n^2 \log^2 n)$ and depth $O(1)$ using constant-depth teleportation. We also need to perform $O(n \log n)$ Hadamards. In addition, we need to perform the following fanouts (and corresponding unfanout). This is illustrated in Figure 3 from [97], which we repeat in Figure 2.5.

- $n' = O(\log n)$ constant-depth fanouts of $2^{n'} = O(n)$ target qubits from each x_i to each R_j register.

- $2^{n'} = O(n)$ constant-depth fanouts of $2^{n'} = O(n)$ target qubits from each S qubit to the R_j registers.

The two kinds of fanouts above take, in total, $O(1)$ depth, $O(n^2)$ size, and $O(n^2)$ width. These are subsumed by the resources needed for reordering qubits above. The operations in the RS module do not require any PAR ancillae qubits to be teleported in.

Second, we consider the T module, into which the $O(\log n)$ input qubits x_i and the $O(n)$ qubits from S are teleported from the RS module. There is a single Hadamard gate on the output qubit and a fanout (and corresponding unfanout) of $O(2^{n'}) = O(n)$ qubits, which takes a depth of $O(1)$ and a size and width of $O(n)$. Finally, we are left with $O(n)$ controlled-R_z rotations of angle $\pi/2^{n-1}$ which can be done in expected depth of $O(1)$, size and width of $O(n)$, and $O(n)$ teleported PAR qubits of the form $(|0\rangle + e^{i\phi_k} |1\rangle)$.

This gives us the required resources stated in the theorem. □

Figure 2.5: A circuit for exact OR_3 from Takahashi-Tani [97].

2.3.4 *A Majority Gate in 2D CCNTCM in Sublogarithmic Depth*

Now we have all the building blocks needed to construct a MAJ_n gate in sublogarithmic depth. Using the constant-depth majority circuits of $O(n)$ fanin from [107] mentioned at the beginning of this section, we will then have a sublogarithmic quantum circuit for modular exponentiation, and therefore for Shor's factoring algorithm.

Theorem 6. ***Constant-depth quantum MAJ_n gate on* 2D CCNTCM.** *A quantum MAJ_n gate can be implemented on 2D CCNTCM with circuit depth $O(1)$, circuit size and width $O(n^2 \log^2 n)$, module depth $O(1)$, and module size and width $O(n \log n)$.*

Proof. We combine the gates Below we give the construction for $\mathrm{MAJ}_n(x)$ on the n-qubit input x.

1. We compute in parallel the gates $\mathrm{EX}_n^i(x)$ for $0 \leq i \leq \lceil n/2 \rceil$ to determine if the quantum threshold for majority is reached. There are at most $(n/2)+1$ such gates. To do this, we need to use Theorem 5. Each gate EX_n^i requires the following steps:

   (a) Compute the constant-depth reduction from EX_n^t to EX_m^t where $m = \lceil \log_2(n+1) \rceil$, using the reduction from OR_n to $\mathrm{OR}_{\log_n}$ [47]. For $1 \leq k \leq m$, do the following:

      i. Compute the qubit $\left| \mu_{\phi_k}^{|x|-t} \right\rangle$, which is the rotation by Hamming weight of x, with a threshold t subtracted, by the angle $\phi_k = 2\pi/2^k$. Note that the required precision of this angle is $O(\log \log n)$. This uses the result of Theorem 3. This can be done by a 2D CCNTCM circuit with $O(1)$-depth, $O(n^2)$-size, and $O(n^2)$-width.

   At the end of this step, we have $m = O(\log_2 n)$ bits $|y_k\rangle$. If $|x| \geq t$ then the qubits $|y_k\rangle$ will be orthogonal to the state $|0^m\rangle$.

(b) Apply the 2D CCNTCM circuit for exact $\mathrm{OR}_{\log n}$ from Theorem 5 of [97] to the output of the previous step. This can be done with a 2D CCNTCM circuit with $O(1)$-depth, $O(n \log n)$-size, and $O()$ width.

At the end of this step, we have used expected circuit depth of $O(1)$, expected circuit size and width of $O(n^3 \log^2 n)$, expected module depth of $O(1)$, and expected module size and width of $O(n \log n)$.

2. Apply the gate $\mathrm{PA}_{\lceil n/2 \rceil}$ to the result of the previous step. This can be done by a 2D CCNTCM circuit of $O(1)$-depth, $O(n)$-size, and $O(n)$-width using constant-depth fanout, as described in Section 0.3, and conjugated by Hadamards on every qubit as described in [74]. See the equivalence of these two steps in Figure 2.6.

Figure 2.6: The equivalence of PA_n and FANOUT_n conjugated by Hadamards.

3. Apply a NOT to the output of the previous step. This final output is the output of the quantum majority gate MAJ_n.

The final resources for a quantum MAJ_n gate is shown in Table 2.2, which subsumes those of the PK-KSV quantum compiler resources from Table 3.5.

$\langle D \rangle$	$O(1)$
$\langle S \rangle$	$O(n^3 \log^2 n)$
$\langle W \rangle$	$O(n^3 \log^2 n)$
$\langle \overline{D} \rangle$	$O(1)$
$\langle \overline{S} \rangle$	$O(n \log n)$
$\langle \overline{W} \rangle$	$O(n \log n)$

Table 2.2: Expected circuit resources for a quantum MAJ_n gate.

□

2.4 Conclusion

In this section, we have contributed a nearest-neighbor factoring architecture with sublogarithmic depth based on majority circuits. To do so, we've combined results from classical threshold circuit complexity and our low-depth quantum architectural techniques from Chapter 1. We discuss the effect of quantum compiling single-qubit rotations to a fixed, finite basis. To that end, we've given a concrete circuit for a quantum majority gate on 2D CCNTCM to fit into a classical, constant-depth majority circuit for factoring. Table 2.3 summarizes the final 2D CCNTCM resources for our sublogarithmic factoring architecture, combining Theorems 1, 2, and 6.

$\langle D \rangle$	$O(\frac{1}{\epsilon}(\log\log n)^2)$
$\langle S \rangle$	$O(\frac{1}{\epsilon} n^{6+2\epsilon} \log^4 n \log\log n)$
$\langle W \rangle$	$O(\frac{1}{\epsilon} n^{6+2\epsilon} \log^2 n)$
$\langle \overline{D} \rangle$	$O(\frac{1}{\epsilon})$
$\langle \overline{S} \rangle$	$O(\frac{1}{\epsilon} n^{4+2\epsilon} \log n)$
$\langle \overline{W} \rangle$	$O(\frac{1}{\epsilon} n^{4+2\epsilon} \log n)$

Table 2.3: Expected circuit resources for a sublogarithmic factoring architecture with time-space tradeoff $\epsilon = (0, 1]$.

Now we conclude with some interesting open questions. Although we are able to compile arbitrary rotations down to CCNTC in $O((\log\log n)^2)$-depth for the resolution needed for factoring, can we reduce this to $O(1)$ depth if we relax our requirements? For example, if we have a finite basis which is fault-tolerant, but which is not fixed; it may vary based on the input size. quantum compiling makes truly constant-depth factoring architecture a challenging open problem. Perhaps we will solve this in Chapter 3, when we discuss quantum compiling on nearest-neighbor architectures. Another open question is: is our factoring architectures now optimal? What is the lower-bound for factoring on CCNTC, versus the $\Theta(1)$ bound for factoring on CCAC? Can we determine the optimality of our factoring architectures using some other time-space product? Perhaps we will this

in Chapter 4, when we discuss quantum circuit coherence.

Chapter 3

QUANTUM COMPILING

Quantum compiling is the approximation of an n-qubit unitary operation using a fixed, finite, universal set of simpler gates. Like quantum architecture, quantum compiling plays an intermediate role between quantum algorithms (which determine the high-level gates in a circuit), and quantum error correction (which determines the fault-tolerant gate set). Quantum compiling helps mitigate one source of error, the difference ϵ between a desired gate and its finite approximation, and consumes its own circuit resources, usually measured in $(1/\epsilon)$. Therefore, it is important to study efficient quantum compiling so that we don't lose any quantum algorithmic speedups. Most interestingly to this dissertation, quantum compiling itself is an algorithm and can be mapped to a low-depth, nearest-neighbor architecture. That is the subject of this chapter.

In Section 3.1, we build upon the background of Section 0.1 to rigorously define the problem of quantum compiling and define useful notation and circuit resources, building upon the primer on circuit bases in Section 0.1. We also discuss variations and subtasks of quantum compiling that are common themes in the literature, as well as their interrelationships. These themes include exact synthesis versus approximation, single-qubit gates versus multi-qubit gates, and deterministic versus randomized compiling procedures.

In Section 3.2, we review the foundational result in this field, the Solovay-Kitaev algorithm, and how it continues to shape quantum compiler research. We also discuss lower bounds on any SK-style approach to quantum compiling. Building upon this, in Section 3.3, we review the large amount of recent literature on single-qubit quantum compiling with constant width. We present the cross-cutting themes of this research and provide an at-a-glance comparison of all known single-qubit quantum compiling methods to date in Section 3.4.

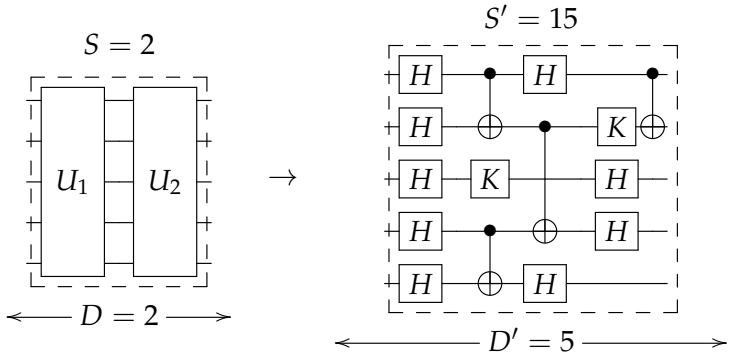

Figure 3.1: An arbitrary quantum circuit being compiled into single-qubit gates and *CNOT*.

As an alternative method of single-qubit quantum compiling, in Section 3.5, we discuss a popular method of trading low depth for increased width: combining phase kickback with a quantum Fourier state.

In Section 3.6, we contribute an improved algorithm for generating quantum Fourier states based on Kitaev-Shen-Vyalyi [53]. We also provide the calculation of parameters necessary for any practical implementation as well as the particular circuit resources consumed on 2D CCNTCM. We then compare the KSV approach to a recent alternative by Jones which distills quantum Fourier states recursively [48]. Finally, in Section 3.7, we contribute a method for quantum compiling the single-qubit rotations needed for a sublogarithmic depth quantum majority gate from Chapter 2.

The last two sections represent the main contributions of this chapter.

3.1 Quantum Compiling Themes

Quantum compiling is a classical procedure for transforming quantum circuits. The image to keep in mind throughout this entire section is shown in Figure 3.1.

General quantum compiling can be subdivided into more special-purpose tasks along several axes, which are cross-cutting themes in any literature review of quantum compil-

ers. These themes also provide a context for understanding the resource consumption for a wide variety of quantum compilers.

These axes are:

1. single-qubit compiling versus multi-qubit compiling

2. exact synthesis versus approximative quantum compiling

3. deterministic versus probabilistic quantum compiling

4. compilers with provable upper bounds versus conjectured upper bounds

The first axis is single-qubit compiling (mentioned previously in Section 0.1.4) versus multi-qubit compiling. Some algorithms which work on single-qubit compiling can be generalized directly to the multi-qubit case. In fact, all known examples of these generalized algorithms can accept an arbitrary circuit basis $\mathcal{B}$ [4, 28, 36, 18]. That is, they do not exploit any special structure of a particular basis. The circuit basis is another input to the algorithm, possibly to an additional classical preprocessing step. Whether the algorithm is a single-qubit or a multi-qubit algorithm depends on whether the basis is single-qubit or multi-qubit.

There is an intermediate point on this axis, between single-qubit and multi-qubit, which is the reduction of a multi-qubit circuit into a basis of single-qubit and two-qubit gates. This task is often called *quantum circuit synthesis*, and we will discuss it in Section 3.1.2.

The second axis is compiling a circuit exactly or approximately. Exact synthesis refers to the case of determining whether a target circuit C is implementable from a basis $\mathcal{B}$ with no error ($\epsilon = 0$). If this is possible, a quantum compiler should return the sequence of gates which constitute the exact synthesis. Furthermore, exact synthesizers often have a goal of returning the *optimal* sequence of compiled gates, that is, one with minimal length ℓ. In the compilers that we review in Section 3.3, ℓ stands for the optimal depth of non-Clifford resources in a basis which also contains Clifford gates. Non-Clifford resources

are always more expensive than Clifford gates in most error-correcting codes. There is evidence that the Clifford resources needed to synthesis the non-Clifford gates T and $Toffoli$ are within a small constant factor of each other [33, 51]

Exact synthesizers often enumerate over all circuits of a certain length from a certain basis $\mathcal{B}$. Therefore, their resources are upper bounded by a brute force search, which takes time upper-bounded by $|\mathcal{B}|^{\ell}$. Approximative quantum compiling conforms to our usual notion where $\epsilon > 0$, and achieving smaller error costs more resources. Many exact synthesis algorithms can be used to build basic approximations for the Solovay-Kitaev algorithm more efficiently, and therefore help achieve better approximative upper bounds as verified by numerical simulation over random unitaries.

What is the relationship between ℓ and ϵ? By a volume argument, the minimum number of points in an ϵ_0-net for $SU(d = 2^n)$ is $1/(\epsilon^{d^2-1})$. If we were to do an approximative search within error ϵ_0 for a circuit in $SU(d)$ which is known to have optimal length ℓ, we would have to enumerate all sequences from a basis $\mathcal{B}$ of up to length ℓ in the worst case, of which there are $|\mathcal{B}|^{\ell}$. Therefore, we have the following relationship.

$$\ell \geq (d^2 - 1) \log_{|B|}(1/\epsilon) + O(1) \tag{3.1}$$

The third axis is whether a quantum compiling algorithm uses randomness or is completely deterministic. For known randomized algorithms, it is an open problem whether the algorithm can be derandomized or not [56], and numerical verification is necessary to show the desired distribution of running times.

The fourth axis is whether a quantum compiler has upper bounds (usually on running time or compiled sequence length) that are provable or based on a conjecture. Both deterministic and randomized algorithms can have provable upper bounds, although in the latter case, one calculates the average-case and upper bounds the variance. Likewise, both deterministic and randomized algorithms can be based on a conjecture. One example is a deterministic algorithm whose resources are too difficult to compute in any other way than numerical simulation and fitting a curve to the data.

These four axes can be used to classify quantum compilers, although some algorithms

can be placed in multiple categories. For example, many single-qubit quantum compilers which perform exact synthesis can be incorporated into a hybrid algorithm which then performs approximation. And of course, some single-qubit quantum compilers can be generalized into multi-qubit algorithms.

A fifth axis could be formed, which is whether the compiled circuit requires arbitrarily long interactions for $CNOT$ or is nearest-neighbor. Such a quantum compiler could also divide up a compiled circuit into an optimal number of modules on 2D CCNTCM to also minimize module depth and module size (inter-module teleportations). This is an interesting direction for future research.

3.1.1 Quantum Compiler Resources

Just as a quantum algorithm with arbitrary long-range interactions incurs some overhead in being mapped to a nearest-neighbor architecture, a quantum compiler itself is an algorithm. It always has a classical component, which runs on a digital computer, and transforms a classical description of an input quantum circuit into an output circuit from a basis $\mathcal{B}$. The compiled output circuit then runs on a quantum computer. In general, the compiled output circuit $\tilde{C}$ consumes resources which are greater than those of the input circuit C.

Not all quantum compilers are "total functions." Some of them, notably single-qubit compilers, are "promise functions" in that they can only compile gates of a certain form (usually $R_Z(\phi)$) and require prior decomposition of a multi-qubit gate down to the set of $Q \cup \{R_Z(\phi)\}$.

classical runtime R**:** the classical time it takes to return a compiled quantum circuit.

input depth D**:** the depth of the input quantum circuit in arbitrary n-qubit gates.

input size S**:** the size of the input quantum circuit in arbitrary n-qubit gates.

input width W**:** the width of the input quantum circuit in qubits.

compiled depth D'**:** the compiled quantum circuit depth, equal to the compiled sequence length for single-qubit circuits.

compiled size S**:** the compiled quantum circuit size, which is identical to compiled depth if no ancillae are used (compiled width is zero).

compiled width W**:** the compiled quantum circuit width, which includes the width of the input circuit as well as any ancillae introduced by the compiler.

All but the first resource are quantum in nature, and follow the definitions for circuit resources from Chapter 1. Because compilation incurs some overhead, we have $D' \geq D$, $S' \geq S$, and $W' \geq W$.

It's also known that in order to approximate a circuit with S gates to a total precision of ϵ requires each gate to be approximated to a precision of $n = O(\log(S/\epsilon)$ [65]. We denote this per-gate precision n, since it serves as an independent parameter for compiling. For single-qubit gates, $S = 1$, and this corresponds exactly with our previous definition for n in Section 0.1.

We do not measure classical space requirements, although these may be exponential. This would be a useful metric for comparison for future work.

3.1.2 Decomposition to Bounded-Qubit Gates

Restricting ourselves to the simplest case of single-qubit circuits allows us to exploit a lot of structure in the group $U(2)$ (or its related subgroups $SU(2)$ and $PSU(2)$). From a volume argument, we can derive a general lower bound for the efficiency of the multi-qubit case [46], as well as determine how our compiling efficiency scales with dimensionality. Any SK-style algorithm produces worst-case sequence lengths ℓ_d that are longer than worst-case single-qubit sequence lengths ℓ_1 by a certain multiplicative prefactor. This prefactor has a dependence that is at least polynomial in $d = 2^n$.

$$\ell_d / \ell_1 = \Omega\left(\frac{d^2 - 1}{\log |\mathcal{B}|}\right) \tag{3.2}$$

This is an example of task modularity which allows us to divide the effort of quantum compiling between the single-qubit case and then decomposition to single-qubit gates and $CNOT$. It is a heuristic which often results in simple decompositions to implement in (classical) software. It may not be asymptotically optimal compared to generic multi-qubit protocols. However, for small input sizes, it is often tractable to run on modern digital computers.

Now that we have handled the single-qubit case, how can we leverage this to compile general n-qubit gates? We need a reduction to the basis $\mathcal{Q} = U(2) \cup \{CNOT\}$, as originally depicted in Figure 3.1. It turns out that almost any two-qubit gate plus arbitrary single-qubit rotations are universal [20]. However, we will stick with CNOT due to its other useful properties. A table of known upper and lower bounds for this task are given in Table 3.1.2. A standard two-level decomposition, such as provided on page 70 of [53], decomposes a general $U(d = 2^n)$ matrix down to $O(d^2)$ "two-level" matrices which can be implemented with multiply-controlled single qubit gates $\Lambda^{n-1}(U)$ for $U \in U(2)$. These gates are implementable with $O(n)$ $CNOT$ gates each.

The recent Giles-Selinger proves a conjecture by Kliuchnikov-Mosca-Maslov [55] that n-qubit circuits implementable by the basis $\mathcal{C}_2 \cup \{T\}$ is equivalent to all $U(2^n)$ matrices with elements from the ring $\mathbb{Z}\left[i, \frac{1}{\sqrt{2}}\right]$. Their construction to find this exact synthesis of an n-qubit gate is meant to prove this equivalent, and is not optimal.

The optimal bound needed for this in terms of $CNOT$ gates in the compiled output (the dominant cost) is still exponential $O(4^n)$ [86].

3.2 The Solovay-Kitaev Algorithm

As one of the central results of quantum computation, it is worth reviewing here the venerable Solovay-Kitaev (SK) algorithm for the approximation of $SU(d)$ gates by a fixed, finite basis $\mathcal{B}$. Although many recent results have surpassed SK in terms of efficiency, they often do so by improving the base-level approximation of the original SK algorithm. Moreover, many techniques for analyzing and understanding quantum compilers were developed for SK, which continues to be the best way to learn them. Finally, the overall

Decomposition Method	*CNOT* Cost
Giles-Selinger [39]	$O(9^n nk)$
Two-level [53]	$O(4^n n)$
Vartiainen-Möttönen-Salomaa [102]	$o(11 \cdot 4^n)$
Aho-Svore [2]	$o(1.17 \cdot 4^n - 3.51 \cdot 4^n + 3.34)$
Shende-Bullock-Markov [85]	$o(0.48 \cdot 4^n - 1.50 \cdot 2^n + 1.34)$
Shende-Bullock-Markov [86]	$\omega(0.25 \cdot 4^n - 3n - 1)$
BBC+ [5]	$\omega(0.10 \cdot 4^n - 0.34n - 0.12)$

Table 3.1: Comparison of multi-qubit circuit synthesizers in *CNOT* cost, both upper and lower bounds. The $o(\cdot)$ and $\omega(\cdot)$ notation are used to indicate when multiplicative constants are known.

structure of the original SK algorithm is so simple, it is surprising that it works so well. Therefore, it is worth our time to review it here.

The essential structure of SK is to recursively generate nets of unitary operators with successively finer precision. At any given level of recursion, the input gate is divided up into two halves which can be approximated with less precision. Our pseudo-code and explanation follows the exposition in [28] and [44]

1: FUNCTION $\tilde{U}_i \leftarrow \text{SK}(U, n)$
2: **if** $i = 0$ **then**
3: $\quad \tilde{U}_i \leftarrow \text{BASIC-APPROX}(U)$
4: **else**
5: $\quad \tilde{U}_{i-1} \leftarrow \text{SK}(U, i-1)$
6: $\quad A, B \leftarrow \text{FACTOR}(U\tilde{U}_{i-1}^\dagger)$
7: $\quad \tilde{A}_{i-1} \leftarrow \text{SK}(A, i-1)$
8: $\quad \tilde{B}_{i-1} \leftarrow \text{SK}(B, i-1)$
9: $\quad \tilde{U}_i \leftarrow \tilde{A}_{i-1}\tilde{B}_{i-1}\tilde{A}_{i-1}^\dagger\tilde{B}_{i-1}^\dagger\tilde{U}_{i-1}$
10: **end if**

11: return $\tilde{U}_i$

Since the recursion must eventually bottom out, we must precompute some sequences of gates from $\mathcal{B}$ up to length l_0. This is the classical preprocessing step which requires upfront storage space for this coarsest-grained net, where each sequence is no more than ϵ_0 from its nearest neighbor. According to [28] the values of $l_0 = 16$ and $\epsilon_0 = 0.14$ using operator norm distance is sufficient for most applications. This step can be done offline and reused across multiple runs of the compiler, assuming $\mathcal{B}$ for your quantum computer doesn't change.

The BASIC-APPROX function above does a lookup (e.g. using some kd-tree search maneuvers through higher-dimensional vector spaces) using this ϵ_0-net, and all higher recursive calls to SK are effectively constructing finer ϵ-nets on the fly as needed.

The FACTOR function performs a balanced group commutator decomposition, $U = ABA^{\dagger}B^{\dagger}$, and then recursively approximates the A and B operators, again using SK. We denote by $\tilde{U}_i$ as the approximation of U using i levels of SK recursion. When they are multiplied together again, along with their inverses, their errors (which go like ϵ) are symmetric and cancel out in such a way that the resulting product U has errors which go like ϵ^2, using the properties of the balanced group commutator. In this manner, we can eventually sharpen our desired error down to any value. A geometric decomposition is used in the Dawson-Nielsen implementation [28], while one based on a Baker-Campbell-Hausdorff approximation is used in the Harrow implementation [44]. It is not known which method converges more quickly to a desired gate in general.

Each level i of recursion solves the problem of compiling U_i by combining five gates compiled at the lower $i-1$ level. Therefore, the compiled circuit size at the top-level is upper-bounded by 5^n, and this is in general the same as the circuit depth (since not all gates in $\mathcal{B}$ commute).

3.3 Quantum Compiler Review

This section reviews all known single-qubit quantum compilers from before 2012 (in Section 3.3.1) to the present (in Section 3.3.2) which use the basis $\mathcal{C}_2 \cup \{T, Toffoli\}$. Quantum

compilers which use alternate, non-Clifford resources are discussed in Section 3.3.4.

3.3.1 Quantum Compiling Before 2012

The first known quantum compiling result by Lloyd showed that almost any two distinct single-qubit gates were universal for single-qubit compiling [65]. However, it could take exponential resources in the precision: $R, D' = O(1/\epsilon) = O(2^n)$ The first *efficient* quantum compiler due to Solovay-Kitaev (SK) as independently implemented by Harrow [44] and Dawson-Nielsen [28] had depth overhead $D' = O(\log^{2.71}(1/\epsilon))$ and classical running time $R = O(\log^{3.97}(1/\epsilon))$. This is a seminal result in the field of quantum computation, and as such it was reviewed in the previous section (3.2). We will abbreviate it afterwards as SK-DN. SK was later improved by Kitaev-Shen-Vyalyi to have a longer running time but smaller depth overhead of $R, D' = O(\log^{3+o(1)}(n/\epsilon))$ [53], an algorithm we will call SK-KSV. This improvement was achieved by repeated rounds of sparsifying and telescoping finer nets based on coarser nets. Both SK-DN and SK-KSV can be generalized to n-qubit compiling, but in the latter case the multiplicative prefactor has a doubly exponentially dependence of $exp(2^n)$. Suggestions for improving this to be $poly(2^n)$ are provided in the text, but this remains an interesting open problem.

In the same text [53], a more parallel compiling procedure was discovered by Kitaev-Shen-Vyalyi with a depth overhead of $D' = O(D\log(1/\epsilon) + (\log\log(1/\epsilon))^2)$ but a size overhead of $S' = O(S\log(1/\epsilon) + (\log(1/\epsilon))^2(\log\log(1/\epsilon))$ and a width overhead of $W' = O(W(\log(1/\epsilon))^2$ on AC. This involves the generation of a circuit-wide resource, an n-qubit quantum Fourier state, which can be amortized over the compilation cost for the entire circuit. Because this algorithm uses phase kickback, we refer to it as An equivalent, alternative proof is given by Cleve-Watrous [26]. PK-KSV was the first quantum compiler to make a time-space (circuit depth versus circuit width) tradeoff, using ancillae to decrease depth. Therefore, it has non-trivial circuit space and circuit width, and it is the first compiler where architectural concerns matter. We calculate detailed parameters and circuit resources for it on 2D CCNTCM in Section 3.6.

As early as 2004, Austin Fowler pioneered the use of optimized enumeration of gate

sequences over a single-qubit basis, using efficient data structures and heuristics to search and de-dupe (remove duplicates from) a database of optimal, unique sequences up to a fixed length [36]. He used the basis $\mathcal{C}_1 \cup \{T\}$ and gave efficient, fault-tolerant implementations of this basis in the Steane code. Due to the fact that the Clifford group is closed under matrix multiplication, every compiled Fowler sequence can be simplified as a Clifford gate alternating with T. Asymptotically, this still runs in time $|\mathcal{B}|^\ell$ to produce compiled sequences of optimal length ℓ, with an improved multiplicative constant. It is an exact-synthesis, deterministic, provably optimal compiler. In its current form, it is a single-qubit compiler because it takes as input a single-qubit basis, but it can be generalized to n-qubits by enumerating and feeding in $\mathcal{C}_n \cup \{T\}$. Unfortunately, the size of the Clifford group increases at least exponentially in n, which places practical limits on any such enumerative compiler for $n > 3$.

In 2007, Matsumoto-Amano used the idea of a *normalized circuit* to improve the enumeration and search approach to compiling [68]. Two circuits in normalized form compute the same gate if and only if they are the same circuit. If a normalized form can be computed efficiently, then a database of normalized forms can be easily searched to determine the uniqueness of a gate sequence. Their case exploits the special structure of the single-qubit basis $\{H, T, T^\dagger\}$ and is an exact-synthesis, deterministic, provably-optimal compiler over this basis. Both the Fowler and the Matsumoto-Amano compiler can be plugged into Solovay-Kitaev to optimize the base approximations.

3.3.2 Quantum Compiling in 2012 and After

In 2012 and after, quantum compiling experienced a renewed interest in the research community. Many different themes from the previous section were taken up and developed. These developments are summarized below.

In 2012, two approaches improved upon the Fowler method by considering a meet-in-the-middle approach: the work by Amy-Mosca-Maslov-Roetteler (AMMR) [4] and that by Booth ([18]. The Fowler approach searches for an optimal path from identity to the desired target gate using an optimal length of ℓ. The meet-in-the-middle approach divides

this into two tasks of reaching a middle point, searching simultaneously from identity and the target gate, to find optimal paths of length $\lceil \ell/2 \rceil$ and $\lfloor \ell/2 \rfloor$. This gives a quadratic speed improvement over the Fowler approach.

Both the AMMR and Booth compilers can be generalized to multi-qubit circuits by feeding in a multi-qubit basis. In fact, the AMMR compiler was able to generate the two-qubit Clifford group with size $|\mathcal{C}_2| = 11,520$ and the three-qubit Clifford $|\mathcal{C}_3| = 92,897,280$, and therefore it is able to optimize the depth of T gates in a circuit, in addition to optimizing the total count of T gates. However, for multi-qubit gates of $n \geq 4$, the running time of enumerative search, even with optimizations, is doubly exponential (exponential enumeration over an exponential-sized set of Clifford group elements), which quickly encounters practical limits of modern digital computers. On a more positive note, the AMMR compiler gives the first new decomposition of the Toffoli gate in terms of $\mathcal{C}_1 \cup \{T, T^\dagger\}$ since 1995 [5].

The Bocharov-Svore (BS) compiler combines the idea of normalizing circuits and enumerative search to compile *canonical circuits* on single-qubits. Canonical circuits implement the same circuit if their matrices are the same up to conjugation by two (possibly distinct) Clifford gates. This leads to a fourth-root speedup of search time for this exact-synthesis, single-qubit compiler over the $\{H, T, T^\dagger\}$ basis. While its circuit resources can be expressed in terms of basis size $|\mathcal{B}|$ for comparison with the AMMR, Booth, and Fowler compilers, in fact the BS compiler is only known to work for one particular single-qubit basis. Furthermore, the BS exact compiler can be plugged into SK to optimize the enumeration and storage of basic (o-level) approximations. When this is done and simulated for random unitaries, numerical experiments reveal an improved compiled sequence length of $O(\log^{3.4}(1/\epsilon))$. We can call this hybrid compiler SK-BS.

Kliuchnikov-Mosca-Maslov (KMM) provide significant advances for both the exact-synthesis and approximative approaches to compiling, which we can abbreviate KMMe and KMMa, respectively.

For the exact-synthesis case [55], KMMe provides a rigorous proof of the equivalence of circuits over the basis $\mathcal{C}_1 \cup \{T, T^\dagger\}$ and $U(2)$ matrices with elements from the ring

$\mathbb{Z}\left[i, \frac{1}{\sqrt{2}}\right]$. Reducing the problem of circuit synthesis over this basis to state generation, with elements from $\mathbb{Z}\left[i, \frac{1}{\sqrt{2}}\right]$, KMMe remarkably finds sequences of optimal length ℓ in time polynomial in ℓ ($O(\ell^2)$ bit operations), and not exponential in ℓ as all previous approaches. This is a fair comparison because other exact-synthesis algorithms above implicitly depend on a fixed-precision floating-point arithmetic, and KMMe makes its bit-dependence explicit. The KMMe is a single-qubit compiler, but its generalization to higher dimensions depends on proving the conjecture that $U(2^n)$ matrices of elements from $\mathbb{Z}\left[i, \frac{1}{\sqrt{2}}\right]$ are equivalent to n-qubit circuits over the same basis.

For the approximative case [56], KMMa saturates the lower bound of $\Omega(\log(1/\epsilon)$ by using two ancillae prepared in the $|00\rangle$ state and basis of $\mathcal{C}_1 \cup \{T\}$. It reduces the problem of approximating a $R_Z(\phi)$ single-qubit rotation with expressing a large integer M as the sum of four squares. There is a known probabilistic algorithm to solve this Diophantine equation by Rabin-Shallit [78] that takes $O(\log^2(M) \log\log M)$. Therefore, the KMMa compiler runs in classical time $O(\log^2(1/\epsilon) \log\log(1/\epsilon))$ and returns sequences of length $O(\log(1/\epsilon)$, which is optimal up to a constant factor.

A natural improvement is to see if the ancillae of KMMa can be done away with completely. This question is answered in the affirmative by a single-qubit, approximative compiler by Selinger [84]. In that work, general unitaries are approximated up to error ϵ via the $R_Z(\phi)$ (or equivalently, $R_X(\phi)$ rotations) of accuracy $\epsilon/3$ using elements of the ring $\mathcal{R}$ so that a known, optimal exact-synthesizer such as KMMe can be used. The approximation to $R_Z(\phi)$ involves solving Diophantine equations to find approximations to real numbers in the ring $\mathbb{Z}\left[\sqrt{2}\right]$. Solutions to this require random sampling of candidate solutions from $\mathcal{R}$ whose average success probability depends on an open conjecture about the distribution of odd primes. The running time and optimal sequence-depth of the Selinger compiler is verified by numerical simulation.

Improving SK-style approximative and Fowler-style enumerative compilers has dominated most of the progress in the years 2012-2013. However, we now mention a work which uniquely improves on phase-kickback and the PK-KSV-style of compilation that allows $O(n)$ ancillae in order to achieve $o(\log(1/\epsilon)$ depth.

Based on previous multilevel distillation protocols for magic states [51], Jones devised a remarkable way to recursively distill Fourier states using an initial approximation of 2-qubit states using only Clifford gates. This provides an alternative to generating quantum Fourier states to PK-KSV to be used with phase kickback; therefore we abbreviate this method PK-Jones. Fourier state distillation results in an asymptotic improvement in the number of non-Clifford (Toffoli) gates required versus PK-KSV ($O(n \log n)$ versus $O(n^2 \log n)$ for an n-qubit Fourier state), while maintaining the same asymptotic depth overhead ($O(\log n)$). The resources of PK-Jones are compared with an optimized version of PK-KSV in Section 3.6.

All of these approaches, which admit direct comparison more-or-less, are displayed in Table 3.2 in Section 3.4.

3.3.3 Magic States for Quantum Compiling

Instead of using non-Clifford gates to complete our basis into a universal gate set, we can include "non-Clifford" states as an additional resource. These states are called *magic states* if, combined with the Clifford gates, enable universal quantum computation. Quantum compilers which use these magic states count them as a non-Clifford resource instead of T or Toffoli gates, since such states cannot be produced directly using Clifford gates. Rather, they must be distilled in a probabilistic procedure which itself uses only Clifford gates and measurement.

There are two types of magic states, The most famous such example is the $+1$ eigenstate of the Hadamard operator, often written as:

$$|H\rangle = \cos(\pi/8)\,|0\rangle + \sin(\pi/8)\,|1\rangle \tag{3.3}$$

Magic state distillation was originally proposed by Bravyi and Kitaev [19] as a model of universal quantum computation (UQC) which allowed for noisy, or imperfectly prepared, initial quantum states. These noisy states could be "purified" or distilled down to certain states that are "magic" in that, combined with the Clifford gates, they enable UQC. These magic states come in two types, the eigenstates of the H or another operator,

which we call A.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad A = e^{i\pi/4} SH = \frac{e^{i\pi/4}}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ i & -i \end{bmatrix} \tag{3.4}$$

The noisy initial states can be considered to be prepared These states are distilled via successive rounds of recursive error correction, where the efficiency of distillation with each round, and the resources required to do so, depend on the code used.

3.3.4 Alternative Bases and Resources

Now we turn to three recent works which use magic state distillation to compile arbitrary single-qubit gates. These works either compile to an alternate basis from $\mathcal{C}_n \cup \{T, T^\dagger\}$.

A recent approach by Bocharov-Gurevich-Svore [15] compiles to subsets of the Clifford group augmented with the non-Clifford V-basis, first discovered by Lubotsky-Phillips-Sarnak [66], which was proven to permit the lower-bound of compiled sequence length $O(\log^1(1/\epsilon)$ [46].

$$V_1 = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 & 2i \\ 2i & 1 \end{bmatrix} \qquad V_2 = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix} \qquad V_3 = \frac{1}{\sqrt{5}} \begin{bmatrix} 1+2i & 0 \\ 0 & 1-2i \end{bmatrix} \tag{3.5}$$

This work uses the properties of Lipschitz quaternions with norms 5^l, ($l \in \mathbb{Z}, l \geq 0$). It contains a randomized algorithm whose running time is based on a conjecture from geometric number theory. There is currently no complete, fault-tolerant method of compiling all three gates from the V basis into our usual universal set of $\mathcal{C}_1 \cup \{T\}$. However, the appendix of [15] gives a method for implementing the exact V_2 gate using the (probabilistic) magic state distillation of Duclos-Cianci and Svore [32]. The average cost in $|H\rangle$ states for enacting a V_2 gate is 22.75. It is an open problem how many $|H\rangle$ states are needed to enact the gates V_1 and V_3. We cannot compare it directly to previous algorithms which consider the number of T gates (T_c) the primary resource, or the compiled sequence length D' as an upper bound to T_c. However, the SK algorithm allows us to provide an alternate basis. By comparing the compiled sequence lengths in the V basis and noting any improvements over the $\mathcal{C}_1 \cup \{T\}$ basis, measuring the length of compiled sequences

The previous works all assume a minimal basis which only includes a fixed number (usually one) non-Clifford gate (a T gate or Toffoli). We can call these "reduced instruction set computing" (RISC) bases, in analogy to digital processor architectures. In contrast, Landahl and Cesare consider a "complex instruction set computing" (CISC) basis [60] which can include certain non-Clifford states of the form $|0\rangle + e^{i\pi/2^k}|1\rangle = R_Z(\pi/2^k)|+\rangle$. These non-Clifford states can be distilled to a certain error ϵ' using a probabilistic procedure using only Clifford gates similar to distilling the state $|H\rangle$ described above. This error ϵ' is proportional to the desired error of gate compilation: $\epsilon'\tilde{\epsilon}$. Therefore, the work of Landahl-Cesare also unifies into the sum of the exponent of the logarithm $\alpha + \beta + \gamma$ the normally distinct procedures of magic state distillation (α), quantum compiling (β), and error-correction (γ) into the sum of exponents of the logarithm.

$$R, D = O(\log^{\alpha+\beta+\gamma}(1/\epsilon) \tag{3.6}$$

By optimizing over magic state distillation and quantum compiling as a single task, Landahl-Cesare achieves $\alpha + \beta \approx 1$, when the best that can be achieved with these two procedures separately is $\alpha + beta \geq 2$.

3.4 Quantum Compiler Comparison

Algorithm	A/E	D/R	P/C	Multi?	R	D'	S'	W'
Fowler[36]	A	D	P	Y	$O(\ell\|\mathcal{B}\|^{\ell})$			
Booth [18]	A	D	P	Y	$O(\ell\|\mathcal{B}\|^{\ell/2})$			1
AMMR [4]	E	D	P	Y	$O(\ell\|\mathcal{B}\|^{\ell/2})$			1
BS [16]	E	D	P	N	$O(\ell\|\mathcal{B}\|^{\ell/4})$			1
KMM-e[55]	E	D	P	?	ℓ^2			1
SK-DN[28]	A	D	P	Y	$O(n^{2.71} + \mathcal{B}^{l_0})$	$O(Dn^{3.97})$		1
SK-KSV[53]	A	D	P	N	$O(n^{3+o(1)})$	$O(Dn^{3+\nu})$		1
SK-BS [16]	A	D	C	N	$O(n^{2.71} + \mathcal{B}^{l_0})$	$O(Dn^{3.4})$		1
KMM-a[56]	A	P	C	N	$O(n^2 \log n)$	$O(Dn)$		1
Selinger[84]	A	P	C	N	$O(n^4)$	$D(48n + 44)$		1
PK-KSV [53, 26]	E	D	P	N	$O(1)$	$O(D \log n + \log^2 n)$	$O(Sn + n^2 \log n)$	$O(n^2)$
PK-Jones [48]	E	D	P	N	$O(1)$	$O(D \log n + n \log n)$	$O(Sn + n \log n)$	$2n + O(1)$

Table 3.2: A comparison of single-qubit quantum compilers, where $n = \log_2(1/\epsilon)$, for a desired error ϵ. Blank depths D' are ℓ. Blank sizes S' are equal to D'. Blank widths W' are equal to 1. A/E: Approximate versus exact-synthesis. D/R: Deterministic versus randomized. P/C: provable bounds versus conjecture + numerical verification. Multi?: Y if it can be generalized to a multi-qubit compiler.

3.5 *Phase Kickback and Quantum Fourier States*

In this section, we discuss the *phase kickback* procedure for producing a qubit with arbitrary phase: $|0\rangle + e^{i\phi}|1\rangle$. From the PAR procedure, a qubit could be prepared in such a state "offline' and then teleported later into a 2D CCNTCM module to probabilistically enact rotations $R_Z(\phi)$ [51].

Phase kickback requires controlled addition on a target state in the quantum Fourier basis. This will provide us the foundation to understand the KSV procedure for generating such quantum Fourier states in the next section.

In Section 3.5.1, we discuss the properties of the basis of quantum Fourier states, including their relationship with addition modulo 2^n. In Section 3.5.2, we calculate circuit resources for performing these adders on 2D CCNTCM. Although the adders are generic, they will be used in the resource calculations of KSV phase estimation in Section 3.6.3, which operates on quantum Fourier states.

3.5.1 *Properties of the Quantum Fourier Basis*

The following states are well-known as n-qubit quantum Fourier states, the result of applying the quantum Fourier transform (QFT) to the n-qubit computational basis state.

$$\left|\psi_n^{(k)}\right\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} e^{-2\pi i j k/2^n} |j\rangle \tag{3.7}$$

These states form an alternate, orthonormal basis indexed by $0 \leq k < 2^n$. We will often just call these Fourier states, and neglect the subscript n, which is implied. Note that the state $\left|\psi^{(0)}\right\rangle$, sometimes called the fundamental Fourier state, is simply the equal superposition of all computational basis states, or the tensor product of n qubits in the state $|+\rangle$, or the result of applying n Hadamards qubit-wise to a state beginning in $|0\rangle^n$.

Note that these states $|\psi\rangle_n^{(k)}$ are the QFT states of the n-qubit computational basis. The usual method of creating these states involves performing phase estimation of the modular addition operator. These are implicitly hard in that all known procedures take size $O(n \log n)$, even if the depths can be decreased to $O(\log n)$ [51] or in some cases $O(1)$

given unbounded quantum fanout [22].

However, we can create a superposition in constant depth over all odd $k = (2s-1)$ by starting in the state $|0\rangle^{\otimes n}$, then applying a Hadamard and σ^z to the most significant qubit.

$$|\eta\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}\left|2^{n-1}\right\rangle = \frac{1}{\sqrt{2^{n-1}}}\sum_{s=1}^{2^{n-1}}|\psi_{n,2s-1}\rangle \tag{3.8}$$

More obviously relevant to our overall goal of approximating $\Lambda(e^{i\phi})$, we can enact a phase shift simply by performing the following modular addition operator, for which $|\psi_{n,k}\rangle$ are eigenstates.

$$A|j\rangle \to |j+1 \bmod 2^n\rangle \tag{3.9}$$

Applying this operator to its eigenstates results in a phase shift which depends on the particular eigenstate.

$$A|\psi_{n,k}\rangle = e^{2\pi i\phi_k}|\psi_{n,k}\rangle \tag{3.10}$$

Finding the eigenvalue $e^{2\pi i\phi_k}$ corresponds to finding the phase $\phi_k = k/2^n$. Repeated application of A (say p times) would result in a phase added to the eigenstate equal to a multiple of $e^{2\pi ip/2^n}$

$$A^p|\psi_{n,k}\rangle = e^{2\pi i\phi_k/2^n}|\psi_{n,k}\rangle \tag{3.11}$$

This explains why we don't find even k interesting, since then we would not get a cyclic distribution of 2^n different phases, since only odd k are coprime with 2^n. The exception is $k = 0$, since this is the equal superposition of computational basis states, which we can also efficiently create. This will be a useful starting point later on to create addition eigenstates for odd k.

$$|\psi_{n,0}\rangle = H^{\otimes n}|0^n\rangle$$

Suppose we have a certain state $|\psi_{n,k}\rangle$ but we want to get enact a phase shift $e^{2\pi il/2^n}$. We can do this by solving $p = p(s,l)$ in this equation:

$$(2s-1)p \equiv l (\mathrm{mod}\, 2^n) \tag{3.12}$$

Stipulating k to be odd guarantees that there is a unique solution p.

We then apply A^p as follows, where $\mathrm{Y}_n(A)$ means to apply A to the second register p times, where p is an n-qubit number in the first register.

$$\mathrm{Y}_n(A) |p\rangle |\psi_{n,k}\rangle \to e^{2\pi il/2^n} |p\rangle |\psi_{n,k}\rangle \tag{3.13}$$

If we control the operation of $\mathrm{Y}_n(A)$ on a source qubit $|+\rangle$, it will acquire the phase $e^{2\pi il}$.

$$\Lambda(\mathrm{Y}_n(A)) |+\rangle |p\rangle |\psi\rangle^{(k)} \to \left(|0\rangle + e^{2\pi il} |1\rangle\right) |p\rangle |\psi\rangle^{(k)} \tag{3.14}$$

This is not quite the phase kickback procedure, since we must still solve for p using the equation below by finding the modular inverse $(2s-1)^{-1} \bmod 2^n$, making use of the following expansion from Section 13 of [53]

$$p \equiv -l \sum_{j=0}^{m-1} (2s)^j \equiv -l \prod_{r=1}^{t-1} \left(1 + (2s)^{2^r}\right) \mod 2^n \tag{3.15}$$

where $m = O(n)$ is n rounded to the nearest power of 2. In general, this requires a circuit of size $O(n^2 \log n)$ and depth $O((\log n)^2)$ and represents the most expensive part of the KSV procedure as originally presented in [53].

Ideally, we would obviate the need for the expensive circuit above by ensuring that $k = 1$, in which case $p = l$. We will see how to do this in Section 3.6.3.

Finally, to copy the state $|\psi_{n,k}\rangle$ it suffices to apply the following operator which only uses subtraction (addition with one addend and the outcome negated in two's complement representation).

$$|\psi_{n,k}\rangle^{\otimes m} = W^{-1} \left(|\psi_{n,0}\rangle^{\otimes} (m-1) \otimes |\psi_{n,k}\rangle\right)$$

where W is the operator on m registers, each of consisting of n qubits, which adds all the registers into its final register (modulo 2^n).

$$W : |x_1, \ldots, x_{m-1}, x_m\rangle \rightarrow$$
$$|x_1, \ldots, x_{m-1}, x_1 + \ldots + x_m \bmod 2^n\rangle \quad (3.16)$$

Having complained about the cost of modular division, how can we now implement the operators A and W more efficiently on our nearest-neighbor architecture?

3.5.2 *Circuit Resources for Adders on Quantum Fourier States*

First we prove a general result for mapping any AC circuit to a 2D CCNTCM circuit. Then we use it to map the operator A (increment modulo 2^n) and the operator W (multiple addition modulo 2^n) to 2D CCNTCM.

Lemma 2. *Suppose an AC circuit $\mathcal{C}$ has depth $D(n)$, size $S(n)$, and width $W(n)$. Then it can be mapped onto a 2D CCNTCM circuit $\mathcal{C}'$ with circuit depth $D(n)$, circuit size $S(n)$, circuit width $D(n) \cdot W(n)$, module depth $D(n)$, module size $D(n) \cdot W(n)$, and module width $D(n)$.*

Proof. Each of $D(n)$ layers in $\mathcal{C}$ could operate on non-nearest-neighbors. We teleport all $W(n)$ qubits from the first module to the last module in sequence, reordering them at each module so that we can execute the gates of that layer only on nearest-neighbor qubits. This gives the desired circuit and module resources on 2D CCNTCM. □

Lemma 3. *The operator $A|j\rangle \rightarrow |j + 1 \bmod 2^n\rangle$ on n-qubit register can be implemented in circuit/module depth $O(\log n)$, circuit size $O(n)$, circuit width $O(n \log n)$, module size of $O(n \log n)$, and module width of $O(\log n)$ on 2D CCNTCM.*

Proof. We modify the QCLA adder from Section 1.7.1 mapped to 2D CCNTCM using Lemma 2. To add $2 \times n$-bit numbers, the original AC adder has depth $O(\log n)$ and size and width of $O(n)$. Therefore, we have the desired resources for a 2D CCNTCM circuit. □

Lemma 4. *The operator $W|x_1\rangle|x_2\rangle \cdots |x_m\rangle \rightarrow |x_1\rangle|x_2\rangle \cdots |x_1 + x_2 + \ldots + x_m \bmod 2^n\rangle$, which operates on $m \times n$-qubit registers, can be implemented in circuit/module depth $O(\log n + \log nm)$,*

circuit size $O(mn)$, circuit width $O(mn + n \log n)$, module size of $O(mn + n \log n)$, and module width of $O(m + \log n)$ on 2D CCNTCM.

Proof. This reduces to the problem of modular multiple addition from Section 1.6.3, with the following variation. Instead of addition modulo $m < 2^n$, we perform addition modulo 2^n, which is much simpler. It does not involve any truncation and adding back of modular residues. Namely, we just perform one round of constant-depth $3 \to 2$ addition, then we redo the computation of the highest-order bit v_n to uncompute it.

Therefore, we have a $O(\log m)$-depth binary tree of $O(m)$ modules, which requires a total of $O(mn)$ teleportation of qubits between them. Within each module, the addition takes depth $O(1)$ and size and width of $O(n)$. Combine this in series with the circuit for A in Lemma 3 and we have the desired resource bounds. □

In fact, we also provide numerical upper bounds for circuit resources of both A and W in Table 3.3.

Operation	D	S	
A: CSE Addition of $3 \times n$-bit Numbers Modulo 2^n	66	$56n$	
Q: QCLA Adder from CSE	$4 \log_2 n + 6$	$10n - 48 \log_2 n - 78$	$8n$ lo
W: Multiple CSE Addition of $t \times n$-bit Numbers Modulo 2^n	$57 \log_2 t$	$100nt - 200n$	$5n$

Table 3.3: Circuit resources for adders used in QFS generation on 2D CCNTCM.

Armed with these properties, we're now ready to describe our version of the KSV phase estimation procedure to product quantum Fourier states.

3.6 Circuit Resources for the KSV Quantum Compiler

This section gives a pedagogical review of the quantum compiling method that combines phase kickback and QFS generation due to Kitaev-Shen-Vyalyi [53]. In this form, it compiles single-qubit of the form $R_Z(\phi)$. Using a multi-qubit decomposition such as those presented in Section 3.1.2, it can be extended to compile multi-qubit gates. Furthermore,

we present an original optimization called *early measurement* which does not asymptotically increase our compilation resources (see Appendix A). We refer to our optimized method as PK-KSVe, which still shares much in common with PK-KSV up to the phase estimation step. Finally, we contribute the architectural resources on 2D CCNTCM for running three different QFS generation.

First, in Section 3.6.1 we discuss our optimization of early measurement, which distinguished PK-KSVe from the original PK-KSV algorithm. Next, in Section 3.6.2 we present the high-level overview of the KSV algorithm. In Section 3.6.3 the most important and most resource-intensive step in PK-KSV and PK-KSVe. Section 3.6.4 describes the classical post-processing which completes phase estimation. While this is done on a classical computer, we will discuss how this step affects our parameters and resources for generating a quantum circuit. We also discuss a new, faster phase estimation algorithm [94]. Finally, we map the QFS stage of PK-KSV, PK-KSVe, and PK-Jones to 2D CCNTCM and compare their asymptotic resource usage in Section 3.6.5.

3.6.1 The Optimization of Early Measurement

In this section, we describe the differences between PK-KSV, as originally presented in Section 13 of [53] and PK-KSVe, which includes our novel contribution of early measurement.

Given a circuit C to compile, we precompile it into gates from $\mathcal{G} \cup \{R_Z(2\pi x/2^n)\}$ using the results from Sections 0.1.4 and 3.1.2 in $O(1)$ time, depth, and size. Now we are done with the single-qubit gates and CNOT, and we have computed the values $\{x_1, \ldots, x_m\}$ that allow us to approximate our desired m $R_Z(\phi)$ gates as $\phi \approx \frac{\pi l}{2^n}$ to within precision 2^{-n}. We now need to generate the QFS $\left|\psi^{(k)}\right\rangle$ to use with phase kickback to either approximate $R_Z(\phi)$ directly, or to generate PAR qubits as intermediaries for later enacting $R_Z(\phi)$ probabilistically.

The original PK-KSV procedure first created the state $\left|\psi^{(0)}\right\rangle$ and applied phase estimation to it to enact the operator $\mathrm{Y}(e^{-2\pi i/2^n})$ and get $\left|\psi^{(1)}\right\rangle$ as a result. This state can be copied using W: multiple addition modulo 2^n. However, it involves using coher-

ent measurement in order to enact a phase $|x\rangle \rightarrow e^{2\pi i x/2^n}|x\rangle$ on all the components of $\left|\psi^{(0)}\right\rangle = \sum_{x=0}^{2^n}|x\rangle$.

The PK-KSVe does not enact the operator $\mathrm{Y}(e^{-2\pi i/2^n})$ but instead yields a QFS $|\psi(k)\rangle$ for random odd k, as well as the index k itself. This can be used to solve the modular inverse equation classically:

$$kp \equiv x \bmod 2^n \tag{3.17}$$

to get the solution $p = p(k,x)$. This integer p is added to $\left|\psi^{(k)}\right\rangle$, controlled on some qubit $|phi\rangle$ to get the desired phase shift: $R_Z(2\pi x/2^n)|\phi\rangle$. To copy $\left|\psi^{(k)}\right\rangle$, consider the controlled addition operator $\mathrm{Y}(A)_n$ on two n-qubit registers.

$$\mathrm{Y}(A)_n|x\rangle|y\rangle \rightarrow |x\rangle|y + x \bmod 2^n\rangle \tag{3.18}$$

Applying this operator to two QFS's has the following interesting property as noted in [48].

$$\mathrm{Y}(A)_n\left|\psi^{(k')}\right\rangle\left|\psi^{(k)}\right\rangle \rightarrow \left|\psi^{(k'-k)}\right\rangle\left|\psi^{(k)}\right\rangle \tag{3.19}$$

Therefore, we can copy a state $\left|\psi^{(k)}\right\rangle$ in the second register by putting $\left|\psi^{(0)}\right\rangle$ in the first register to get $\left|\psi^{(-k)}\right\rangle = \left|\psi^{(2^n-k)}\right\rangle$. This is not exactly the same state as $\left|\psi^{(k)}\right\rangle$, but it can similarly be used with solving modular inverse to get an x' and enact a desired rotation $R_Z(2\pi x'/2^n)$.

3.6.2 PK-KSVe Steps

Here now are the main steps of PK-KSVe.

The first stage involves QFS generation.

1. Create the equal superposition of all $\left|\psi^{(k)}\right\rangle$ for odd k:

$$|nu\rangle = \frac{1}{\sqrt{2}}\left(\left|2^{n-1}\right\rangle + |0\rangle\right) \tag{3.20}$$

   in register 1.

2. Use phase estimation with error $\epsilon = 2^{-3n}$ and early measurement to get classical Bernoulli series outcomes y in register 2 and a corresponding QFS $\left|\psi^{(k)}\right\rangle$. This takes a circuit of depth $O(\log n)$ and size $O(n^2)$ (see Section 3.6.3).

3. Perform classical post-processing from Section 3.6.4 to recover the phase k by which can identify the state $\left|\psi^{(k)}\right\rangle$.

Now we have $\left|\psi^{(k)}\right\rangle$, or a state that has high overlap with it (calculated in Section 3.6.5). We can copy it m times using multiple addition W in depth $O(m \log n)$ and size $O(mn)$. This is part of any complete quantum compiler solution for PK-KSVe, but we omit it here and focus on QFS generation.

3.6.3 Parallelized Phase Estimation

Parallelized phase estimation is the central component of both PK-KSV and PK-KSVe which distinguishes it from all other quantum compiling approaches. It is used to randomly "pick" an eigenstate $|\psi_k\rangle$ of a unitary operator U and measure its corresponding eigenvalue (phase) ϕ_k with some degree of precision $\delta = 2^{-n}$ and error probability $\epsilon = 2^{-l}$. As n increases, the phases generally become closer together, which is why we need exponential precision to distinguish between them.

$$U|\psi_k\rangle = e^{2\pi i \phi_k}|\psi_k\rangle$$

Phase estimation holds some superposition of eigenstates $\sum_i \alpha_i |\psi_i\rangle$ in an n-qubit target register, to which it applies repeated measuring operators $\Lambda(U^{2^k})$ controlled on some t-qubit register, which holds an approximation $\tilde{\phi}$ to the real phase ϕ. The unitary U is applied in successive powers of two to get power-of-two multiples of the phase for increased precision. The error probability of approximating the phase to within a given precision is given by the following:

$$\Pr\left[|\phi - \tilde{\phi}| \geq \delta\right] \leq \epsilon$$

The parameter $t = t(\delta, \epsilon)$ encodes the dependence of the number of $\Lambda(U)$ measuring operators as a function of our desired δ and ϵ. It varies according to the exact phase estimation procedure used.

The popular version of phase estimation presented in [76], requires t repeated controlled applications of some unitary U (and its successive powers as U^{2^k}, $0 < k < 2^t$) to a target state which holds some superposition of its eigenvectors, controlled by t bits which will hold the approximation to a corresponding eigenvalue (phase). This version requires applying an inverse quantum Fourier transform (QFT).

To achieve our desired low-depth, we can "parallelize" the application of $\Lambda(U)$ by interpreting the $t = (n+2)s$ control bits as an n-bit number q and apply $\mathrm{Y}(A)\,|q\rangle$ only once. Recall that A is the addition operator on an n qubit target register containing $\left|\psi^{(k)}\right\rangle$, so we can only effectively add the lowest n bits of q. Furthermore, the eigenvalues of A are rational with a fixed denominator, $\phi_k = k/2^n$. To avoid the inverse QFT, which often has small rotations of the form $R_Z(\phi)$, we can do a classical post-processing step as described in Section 3.6.4.

The main steps in parallelized phase estimation as applied to PK-KSV are as follows. As our most important input parameter, we calculate t, the number of measurements that we need (and the number of control qubits that we have). For QFS generation, it suffices to determine the phase with resolution $\frac{1}{2^{n+2}}$, where each bit of the phase is determined with a series of Bernoulli (biased coin flips) of s trials each. As in the original PK-KSV, we have two sets of measurements, one to measure ϕ as a bias projected on the real axis of the unit circle (called the cosine measurements), and one to perform the same measurements rotated by $\pi/2$ (called the sine measurements). These are illustrated below:

$$\Pr(1|k) = \frac{1 + \sin(2\pi\phi_k)}{2} \qquad \Pr(0|k) = \frac{1 + \cos(2\pi\phi_k)}{2} \tag{3.21}$$

Therefore, we have:

$$t = 2(n+2)s \tag{3.22}$$

1. Begin with a t-qubit "phase" register initialized to $|0\rangle^{\otimes t}$. Recall that we have an n-

qubit "QFS" register initialized to $|\nu\rangle$, the equal superposition of $\left|\psi^{(k)}\right\rangle$ for all odd $0 < k < 2^n$.

2. Place the t-qubit register into an equal superposition by applying n Hadamard gates.

3. Treat t as $2s$ groups of bits, each encoding an $(n+2)$-bit number which we call $|q_i\rangle$. This does not involve any addition or other operation, it just determines how we interpret its measurement outcome after we projectively measure it later.

4. Apply the gate $\mathrm{Y}(A)_{n+2}$ to a target ancilla register $|h_i\rangle$ controlled on $|q_i\rangle$, which we interpret as an $(n+2)$-bit number. The register $|h_1\rangle$ is initialized to all $|0\rangle$'s. This can be done in parallel for all $0 \leq i < 2s$ using constant-depth $3 \to 2$ addition. Now we have $2s$ quantum integers, plus the original register $|\nu\rangle$, that we add down using modular multiple addition in logarithmic depth, including the final addition of a CSE number down to a conventional number using QCLA [31]. We use the adders described in Section 3.5.2 on 2D CCNTCM, uncomputing all other intermediate adder qubits back to $|0\rangle$ so that we are only left with $|\nu\rangle$.

5. Reverse the second step by applying another n Hadamards.

6. Projectively measure all t control qubits in the phase register. These qubits now contain classical 0 or 1 as outcomes of $(n+2)s$ Bernoulli trials. The target $|nu\rangle$ now contains a QFS $\left|\psi^{(k)}\right\rangle$ for some as yet unknown k with high fidelity.

7. Read out these outcomes into our classical controller and perform the post-processing in Section 3.6.4. We get an approximation of ϕ with precision δ and success probability $1 - \epsilon$.

3.6.4 Classical Postprocessing

It is now the point to mention that Kitaev's phase estimation procedure contains a post processing step which is completely classical in character, in that they involve a measurement. If this measurement is projective and the outcomes are completely classical, the

remaining steps can be done on our classical computer, and the results fed back into our quantum algorithm to perform controlled addition (phase kickback). Therefore, as long as we can perform these classical algorithms in polynomial time (which we can), we don't really care about the equivalent circuit size and depth.

The steps of classical postprocessing, which will determine some of the parameters in the earlier, quantum part of phase estimation are as follows.

1. Estimate the phase and its power-of-two multiples $2^j\phi_k$ to some constant, modest precision δ'', where $0 \leq j < (n+2)$. For each j, we apply a series of s measuring operators targeting the state $|\nu\rangle$ controlled on s qubits in the state $(|0\rangle + |1\rangle)/\sqrt{2}$, essentially encoding the $2^j\phi_k$ as a bias in a coin, and flipping the coin s times in a Bernoulli trial, counting the number of 1 outcomes, and using that fraction to approximate the real $2^j\phi_k$.

2. Sharpen our estimate to exponential precision $\frac{1}{2^{n+2}}$ using the $(n+2)$ estimates, each for different bits in the binary expansion of ϕ_k. Multiplication by successive powers-of-two shift these bits up to a fixed position behind the zero in a binary fraction representation, where we can use a finite-automata and a constant number of bits to refine our $O(n)$-length running approximation.

Three things are worth mentioning about the interrelation of the parameters between these two steps. Since our phases all have a denominator of 2^n, there is no need to run the continued fractions algorithm on multiple convergents, as is the case with period-finding in Shor's factoring algorithm. Furthermore, the phases are $1/2^n$ apart, therefore it suffices to approximate the phases to within $1/2^{n+2}$ in order to break ties, which is where our range for j comes from above.

The number of trials s comes from the Chernoff bound:

$$\Pr\left[|s^{-1}\sum_{r=1}^{s} v_r - p_*| \geq \delta''\right] \leq 2e^{-2\delta'^2 s}$$

Setting this equal to the desired error probability ϵ we get

δ''	$1/(2\delta''^2)$	δ'	$1/(2\delta'^2)$
$1/16$	128	0.0019525	131,160
$1/8$	32	0.0078023	8,213
$1/4$	8	0.0310880	517

Table 3.4: Parameters for the number of measurements in PK-KSV.

$$s = \frac{1}{2\delta''^2} \ln \frac{1}{\epsilon}$$

We are actually estimating the values $\cos(2\pi \cdot 2^j \phi_k)$ and $\sin(2\pi \cdot 2^j \phi_k)$, so if we wish to know $2^j \phi_k$ with precision δ'', we actually need to determine the $\cos(\cdot)$ and $\sin(\cdot)$ values with a different precision δ', lower-bounding it with the steepest part of the cosine and sine curves.

$$\delta' = 1 + cos(\pi - \delta'')$$

It is possible to improve this bound by adding a bias angle of $\pi/4$ before measurement, and then subtracting it from the recovered phase angle after classical post-processing. This bias angle is inspired by the SHF phase estimation [94]. Then we have $\delta'' \approx \delta'$.

The factor $\frac{1}{2\delta''^2}$ depends on the constant precision with which we determine our $2^j \phi_k$ values. Since classical time is cheap and quantum gates are expensive, it makes sense to minimize the number of trials s. Table 3.4 shows the corresponding values of $1/(2\delta''^2)$ and δ' as a function of various choices for δ'.

By making our δ' exponentially worse (doubling it) we are only increasing the range of j a linear amount (by one). In general, for $\delta' = \frac{1}{2^l}$, we get a final estimate for $\phi = 2^{m-3}$

Projective measurements are irreversible, and it is not so important that we are left with (classical, unentangled) garbage in our t-qubit ancillae register. After all, we only run phase estimation once to get our initial $\left|\psi^{(k)}\right\rangle$ state. The reason why the authors of [53] go to some care to show that all the classical postprocessing steps can be done in polynomial-size and logarithmic-depth is that these must be done to a quantum state

$\left|\psi^{(0)}\right\rangle$ in order to turn it into $\left|\psi^{(1)}\right\rangle$ in original PK-KSV. However, our new procedure sidesteps this requirement, so we are free to offload this processing to a classical controller, which is available in 2D CCNTCM.

Since we have seen KSV-style phase estimation in Chapter 1, it is important to mention here several differences between applying phase estimation to factoring versus applying it to QFS generation. The first difference is that a constant success probability of $\frac{3}{4}$ is no longer good enough. We would like to drive down our error exponentially low as $\epsilon = 2^{-l}$. Second, the operator whose phase we are estimating is now addition modulo 2^n, which is more efficient in circuit resources than modular multiplication (see Section 3.5.2).

In the procedure above, we measured power-of-two multiples of the phase $2^j\phi_k$ to recover single bits of ϕ_k at a time. However, a remarkable recent result by Svore-Hasting-Freedman (SHF) shows how to use information theoretic and signal processing techniques to improve the number of measurements needed from $O(n^2)$ as in conventional KSV to $O(n \log^* n)$. Their key technique is to select random measurements of this form:

$$(2^{j_1} + 2^{j_2} + \ldots + 2^{j_s})\phi_k \tag{3.23}$$

This would allow us to recover multiple bits at a time. SHF empirically determines that the number of trials s in each Bernoulli series scales as $O(\log n)$ for $n = 1000, 10000$, on the same order of magnitude for running factoring on keys of 2048 to 4096 bits. Furthermore, they extend this approach to multiple rounds. Given that the number of measurements in the final round is $s = O(\log n)$ and that each round requires logarithmically fewer measurements than the previous round, they achieve the iterated-logarithmic depth above. These techniques could be used to improve PK-KSV, but we remain conservative for now until numerical upper bounds can be calculated.

3.6.5 Resource Comparisons

We now map both the KSV and Jones algorithms for QFS generation to 2D CCNTCM and compare their resources. We do not consider the phase kickback part of the quantum compiler, since this controlled addition is the same for both QFS algorithms. Like PK-

Jones [51], we set our error as $\epsilon = \left(\frac{\pi}{2^n}\right)^2$. It is not clear why this error was chosen, since for factoring or other polynomial-sized circuits, an inverse-polynomial error precision is sufficient. However, for the purpose of comparison, we set the same error.

The condition for error for PK-Jones is one minus the overlap between a pure Fourier state $|\psi\rangle_n^{(1)}$ and a distilled Fourier state $|\tilde{\psi}\rangle_n^{(1,m)}$ after m rounds.

$$1 - |\langle\tilde{\psi}_n^{(1,m)}|\psi^{(1)}\rangle|^2 = \epsilon \leq \left(\frac{\pi}{2^n}\right)^2 \tag{3.24}$$

The condition for error in PK-KSV is that an incorrect phase indexed by k is returned, one that does not correspond to the eigenstate $|\psi\rangle^{(1)}$. The target register for phase estimation begins in a superposition of eigenstates $|1\rangle = \sum_{s=1}^{2^{n-1}} \left|\psi_n^{(2s-1)}\right\rangle$, and does not deteriorate since none of the measuring operators mix between the orthogonal decomposition of the QFS basis. We make use of Theorem 10 in Appendix A, where the measured index y corresponds to a subspace $\mathcal{M}_y$ of all Bernoulli series outcomes Δ that would correspond to the phase $k = (2s-1)$, and the state left in $\mathcal{N}$ is a (possibly impure) QFS $\overleftarrow{\leftarrow}^{(\|)}$ which has large overlap with $\leftarrow^{(\|)}$. We take the set of Ω of subspaces to be the odd QFS indices $\{1, 3, 5, \ldots 2^n - 1\}$ of which there are 2^{n-1} elements. Taking our phase estimation error to be $\epsilon\left(\frac{1}{2^{3n-1}}\right)$, we then have the following inequality.

$$1 - |\langle\tilde{\psi}^{(y)}|\psi^{(k)}\rangle| = \sum_{s=1}^{2^{n-1}} \sum_{y\in\Delta: f(j)\neq(2s-1)} \Pr(2s-1|y) \leq |\Omega|\epsilon = \frac{1}{2^{2n}} \leq \left(\frac{\pi}{2^n}\right)^2 \tag{3.25}$$

Several differences exist in resource calculation methods and those of PK-Jones [51]. First, that author calculates expected resources, since the distillation method is probabilistic and involves post-selection. Our resources are worst-case, with still a small probability of failure calculated to match the case of PK-Jones with all successful post-selection. We also compare our improvements with those of the original PK-KSV algorithm, for the QFS generation stage. These are given in Table 3.5.

A useful extension of this work would be to calculate numerical upper bounds and compare the two QFS methods for an application, such as Shor's factoring algorithm for realistic key sizes of 2048-4096 bits.

Resource	PK-KSVe	PK-KSV	PK-Jones
D	$O(\log n)$	$O(\log^2 n)$	$O(\log^2 n)$
S	$O(n^2)$	$O(n^2 \log n)$	$O(n \log n)$
W	$O(n^2)$	$O(n^2)$	$2n + O(1)$
$\overline{D}$	$O(\log n)$	$O(\log^2 n)$	$O(\log n)$
$\overline{S}$	$O(n^2)$	$O(n^2)$	$O(n^2)$
$\overline{W}$	$O(n)$	$O(n)$	$O(n)$

Table 3.5: A comparison of circuit resources for QFS generation for PK-KSVe, PK-KSV, and PK-Jones on 2D CCNTCM for error $\epsilon \leq \left(\frac{\pi}{2^n}\right)^2$.

3.7 *Single-Qubit Rotations for Quantum Majority Gate*

We now present a result delayed from Chapter 2, the quantum compiling procedure which completes the quantum majority gate. Recall from that chapter that within each quantum majority gate, we needed to implement single-qubit rotations of the form $2\pi/m$, where $m = poly(n)$, where n is the input size of the number for factoring. We can augment any quantum majority circuit with quantum compiler modules that produce rotated ancillae of the form $|0\rangle + e^{i\phi}|1\rangle$.

To maintain our sublogarithmic depth, we choose the KSV method for generating quantum Fourier states and combine it with phase kickback (PK-KSV). The precision for quantum compiling is $m = 2^{-n'}$, and the resulting resources for applying PK-KSV on 2D CCNTCM in this case are given in Table 3.6, based on calculations from Section 3.6. Note that these resources are deterministic, since they represent the worst case for any single rotation $R_Z(e^{i\phi})$. To convert from n' to n, we use the relationship $n' = O(\log m) = O(\log n)$. However, we note that it is possible to modify the Jones method of Fourier state distillation to have similar performance.

Given such a factory for producing such ancillae, we can compile the rotations $R_Z(2\pi/m)$, for $m = poly(n)$ as in Theorem 6, by directly using the KSV compiler with parameters $n' = \log n$ as in the previous table.

D'	$O((\log\log n)^2)$
S'	$O()$
W'	$O()$

Table 3.6: Quantum compiling resources for PK-KSV for quantum majority gates in factoring an n-bit number.

We now present a more general result for producing rotations which are rational multiples of $2\pi/m$ using a *finite* basis in constant depth and polynomial size. This is not necessary for our sublogarithmic factoring implementation, since we can always produce our desired angles of $2\pi/m$ directly with our quantum compiler. In fact, no polynomial-size quantum circuit will require more than polynomial precision for compiling single-qubit rotations. However, we present our result in the hopes that it will be useful for other quantum algorithms, perhaps one where we must produce the PAR qubit $|0\rangle + e^{2\pi/m}|1\rangle$ "offline" and then produce the rotation $R_Z(2\pi k/m)$ "online."

Theorem 7. ***Compiling a single-qubit rotation over a non-fixed, finite basis.*** *The single-qubit rotation $R_Z(2\pi k/m)$, where $m = poly(n)$, $k \in \mathbb{Z}_m$, can be implemented in expected depth $O(1)$ and expected size and width $O(k)$ on 2D CCNTCM over the finite (but not fixed) basis $\mathcal{G} \cup \{R_Z(2\pi/m)\}$.*

Proof. We use the quantum parallelism method of Hoyer-Spalek [47], which relies on quantum fanout and unfanout on 2D CCNTCM. Our use of the basis $\mathcal{G} \cup \{R_Z(2\pi/m)\}$ implies that we have access to quantum compiler modules for producing the PAR qubits $|0\rangle + e^{2\pi/m}|1\rangle$. Teleport $O(k)$ such qubits into our current circuit. Our desired rotation of $R_Z(2\pi k/m)$ on a target qubit $|\psi\rangle$ can be produced as k parallel applications of $R_Z(2\pi/m)$, which are already diagonal in the same (computational) basis. Fan out the qubit $|\psi\rangle$ k times, apply the rotation $R_Z(2\pi/m)$ using the PAR procedure, then unfanout the qubits. This requires expected $O(k)$ PAR qubits. □

We note here two possible conjectures for improving the above result. The first would allow us to achieve $O(\log m)$ expected size, expected width, and expected number of

teleported PAR qubits. The second would allow us to compile arbitrary rotations to a basis that is both fixed and finite in constant depth.

Conjecture 1. ***Logarithmic Reduction of Compiling Circuit Size and Width.*** *The size and width of the above circuit depend on whatever additional, finite, set of gates $\{R_Z(\phi_{k_i})\}$ used to augment the usual 2D CCNTCM basis $\mathcal{G}$. Let $\phi_{k_i} = 2\pi k_i/m$, then the size and width of a circuit applying* only *$R_Z(\phi_{k_i})$ are proportional to the order of k_i in $\mathbb{Z}_m$, or equivalently, the number of times we must apply the rotation ϕ_{k_i} in parallel to equal the desired rotation ϕ_k. Suppose we are able to find a Chinese Remainder number system for m, that is, a set of pairwise coprime numbers $\{m_1, \ldots m_t\}$ such that $m = \prod_{i=1}^{t} m_i$, where t and the number of bits needed to encode each m_i are $O(m)$ [107]. The Chinese Remainder representation of k is the set of $(\log_2 m)$-bit numbers $x_i = k \bmod m_i$. Then we conjecture that the finite basis $\mathcal{G} \cup \{R_Z(2\pi x_i/m\}$ satisfies the properties above.*

Conjecture 2. ***Constant Reduction of Compiling Depth.*** *The above bases are finite but still depend on the problem input size n. It may be possible to find a basis that is both fixed and finite that would allow for compiling arbitrary single-qubit rotations in constant depth and polynomial size and width, still to precision $1/poly(n)$. This fixed basis would be "polynomially universal" in that it would be the same for all inputs of any size. We conjecture that products of single-qubit gates in $\mathcal{G}$ which, when diagonalized, represent R_Z rotations of irrational multiples of π, would form such a basis.*

In digital computing, the boundary between architecture and compilers is quite porous and is determined by a processor's instruction set. Architecture studies processor resources to solve an algorithm given a particular instruction set which is fixed in hardware. This instruction set is produced by a compiler, a piece of (low-level) software which transforms over pieces of (high-level) software. This instruction set can change based on which algorithms it allows to solve efficiently as well as which processors it allows to manufacture efficiently as well as which operations it allows humans to understand easily. All of these factors combine to make architecture an art and an engineering discipline rather than merely a science.

Quantum computers make this problem even more difficult due to the nature of a

quantum bit. Because transformations between quantum states vary continuously over the space of unitary matrices with complex coefficients, we can only approximate desired quantum logic gates using a fixed set, given to us by fault-tolerance.

Chapter 4

QUANTUM CIRCUIT COHERENCE

In the first two chapters, we presented low-depth nearest-neighbor architectures for factoring an n-bit number. We improved the depth first to be sublinear and then sublogarithmic, but at a polynomial increase in size and width. This represents a time-space tradeoff which can be upper-bounded by the product of the circuit depth and circuit width. Although the title of this dissertation indicates that it is always beneficial to decrease depth, in experimental implementations, we may be constrained by other real-world resources.

Toward this end, we introduce a new circuit resource called *coherence* which quantifies the amount of error-correction that must be performed to maintain a coherent quantum computing state. This can be analogous to the amount of classical controller time or electrical power that a quantum computing experiment consumes while running an algorithm. We define our new resource in Section 4.1, discuss its relationship to other circuit resources on 2D CCNTCM, and calculate coherence for an example circuit.

Decreasing circuit coherence can increase circuit depth or size, which introduces a new tradeoff for quantifying circuit parallelism. Therefore, we compare it to other (time-space) tradeoffs that are common in the literature, including measurement-based quantum computer (MBQC) in Section 4.2 and the reversible pebble game in Section 4.3. In the last case, we prove a direct relationship to factoring.

Finally, in Section 4.4, we apply coherence to modular multiplication from Shor's factoring algorithm. We present a fascinating conjecture to use pebble game simulations to possible decrease coherence and create an asymptotic upper-bound separation from the depth-width product. Finally, we conclude by presenting configurable-depth factoring circuits that a future experimental architect can use to make the appropriate choice.

4.1 Definition of Circuit Coherence

Usually quantum circuits neglect to draw identity gates. When a bare quantum wire appears, what is meant is that the qubit maintains its coherent state until the next non-identity gate comes along to transform it. However, most quantum circuits are drawn at a logical qubit level, assuming no errors occur and a coherent state is maintained. While so far we have maintained that abstraction in this thesis by studying quantum compiling and quantum architecture independently from quantum error correction, we acknowledge it as an important area for optimization and future study. In this chapter, we peek beneath this abstraction barrier to study the effort to maintain a coherent quantum state at the logical qubit level throughout a circuit by defining a new circuit resource called *circuit coherence*.

First, we will define what we mean by an entangling (or disentangling) gate in Section 4.1.1. Then we will build upon this to define a computational subset of qubits in every timestep, in Section 4.1.2. Finally, in Section 4.1.3, we will use the previous two definitions to provide an algorithm for computing reachability and therefore the computational subset for a circuit. This in turn lets us define the resource circuit coherence and describe its relationship to the other circuit resources: depth, size, and width.

4.1.1 Entangling Gates

An entangled quantum state is one which cannot be expressed as the tensor product of two smaller states. This does not depend on what basis we consider for the smaller states. Using the density operator formalism, we can say that a quantum state over two subsystems A and B is entangled if tracing over one of the subsystem *does not* yield the other subsystem as a reduced density matrix.

$$\rho^{AB} \text{ entangled} \iff \left(\text{tr}_A(\rho^{AB}) \neq \rho^B\right) \wedge \left(\text{tr}_B(\rho^{AB}) \neq \rho^A\right) \tag{4.1}$$

A general density matrix for a state across two subsystems A and B can be written as

$$\rho^{AB} = \sum_{i,i',j,j'} p_{ii'jj'} \left|a_i\right\rangle \left\langle a_{i'}\right| \otimes \left|b_j\right\rangle \left\langle b_{j'}\right| \tag{4.2}$$

where $|a_i\rangle, |a_{i'}\rangle$ are any two states on A and $|b_j\rangle, |b_{j'}\rangle$ are any two states on B.

We review here that the trace is a linear operator which distributes across a general density matrix.

$$\mathrm{tr}\left(\rho^{AB}\right) = \sum_{i,i',j,j'} p_{ii'jj'} \mathrm{tr}\left(|a_i\rangle\langle a_{i'}| \otimes |b_j\rangle\langle b_{j'}|\right) \tag{4.3}$$

A reduced density matrix for a particular term is obtained by tracing out one subsystem.

$$\mathrm{tr}_A\left(|a_i\rangle\langle a_{i'}| \otimes |b_j\rangle\langle b_{j'}|\right) = \mathrm{tr}\left(|a_i\rangle\langle a_{i'}|\right)|b_j\rangle\langle b_{j'}| \tag{4.4}$$

Consider a two-qubit gate E_{uv} on single qubits u and v which exist in a larger system. Without loss of generality, we assume that u is *not* entangled with some other, possibly multi-qubit, state on another set of vertices $L \ni v$. The total state on $u \cup L$ is called $\rho^u \otimes \rho^L$.

We call the action of E_{uv} on this combined state a new state σ^{uL}:

$$\sigma^{uL} = E_{uv}(\rho^u \otimes \rho^L)E_{uv}^\dagger \tag{4.5}$$

We call the gate E_{uv} *entangling* between u and v (and between u and L) given the states ρ^u and ρ^L if the new state after applying E_{uv} is entangled, which corresponds to the condition below.

$$\mathrm{tr}_u(\sigma^{uL}) \neq \rho^L \tag{4.6}$$

More generally, if the above condition is true, we call E_{uv} entangling for any multi-qubit states on $V_1 \ni u$ and $V_2 \ni v$ before the application of E_{uv}.

We call a state ρ^L *self-entangled* if every two qubits $(u, v) \in L$ are entangled, but no qubits are entangled with qubits not in L.

Now consider a slightly different setting, where L contains both u and v in a larger self-entangled state ρ^L. We denote by σ^L the resulting state after applying E_{uv} to ρ^L.

$$\sigma^L = E_{uv}(\rho^L)E_{uv}^\dagger \tag{4.7}$$

We call the E_{uv} *disentangling* between u and L (without loss of generality) if after its action on the state ρ^L it is separable into the product state $\sigma^L = \sigma^u \otimes \sigma^{L-\{u\}}$, which corresponds to the following condition:

$$\mathrm{tr}_u(\sigma^L) = \sigma^{L-\{u\}} \tag{4.8}$$

More generally, we can say the E_{uv} is disentangling for any two larger subsystems $V_1 \ni u$ and $V_2 \ni v$, if it is disentangling for any pairs of qubits (u', v') such that $u' \in V_1$ and $v' \in V_2$ for a particular state $\rho^{V_1 V_2}$. Operationally, it is usually more difficult to show that a gate is disentangling.

Given these definitions, a gate E_{uv} that is entangling in the forward time direction on input state ρ is disentangling in the backward time direction ($E_{uv}^\dagger$) on output state $\sigma = E_{uv}^\dagger \rho E_{uv}$. When we do not specify a time direction, an entangling gate E_{uv} is entangling in the forward direction and disentangling in the backward direction, and vice versa for a disentangling gate.

This definition for entangling or disentangling quantum gates depends on knowing the actual states before these gates are applied. Therefore, the entangling or disentangling nature of a gate may not be apparent just by examining a circuit locally, but may require simulation of the entire circuit. This gives an operational definition for identifying entangling/disentangling quantum gates, but it does not give a compact, theoretical description that can be applied generically. This is currently a drawback of our definition, especially for characterizing the behavior of new quantum algorithms that are not yet well-studied.

However, a quantum algorithm designer able to specify a circuit in terms of single-qubit and two-qubit gates often knows when gates are entangling or disentangling. Moreover, the algorithm designer knows which qubits are garbage and that reversing part of a quantum circuit will uncompute these garbage ancillae back to $|0\rangle$. This is the case for quantum circuits in a certain layered form which we describe in Section 4.3, of which the well-known QFT and factoring circuits are special cases. As an overestimate of circuit coherence, we can also consider all two-qubit gates entangling in the worst case with only

single-qubit projective measurement considered as disentangling. However, this will usually not give an upper-bound separation between coherence and the depth-width product. We only consider two-qubit gates that are either potentially entangling or disentangling for some input states; all other two-qubit gates are a tensor product of single-qubit gates and will be treated as such.

4.1.2 Reachability and Computational Subsets

We refer back to our definition of a quantum circuit on CCNTC, which is represented by a graph $G = (V, E)$, with input qubits $I \subseteq V$ and output qubits $O \subseteq V$, and a classical controller. In particular, the set of all qubits is V, and its size is $|V| = W$, the circuit width. Our notion of circuit coherence will not depend on the modules defined in CCNTCM.

Definition 6. ***Entangling Paths.*** *We denote by $E_{uv}^{(i)}$ a two-qubit gate which acts in timestep i which is either entangling or disentangling for its current states on $|u\rangle$ and $|v\rangle$. An* entangling path *of gates from qubit u in timestep i_1 to qubit v in timestep i_n is any sequence of entangling gates $(E^{(i_1)}, E^{(i_2)}, \ldots, E^{(i_n)})$ where the following conditions are met:*

1. *$E^{(i_1)}$ operates on qubit u and $E^{(i_n)}$ operates on qubit v.*
2. *any two consecutive gates in the sequence $(E^{(i_j)}, E^{(i_{j+1})})$ act on a common qubit w.*
3. *any two consecutive gates in the sequence either occur in consecutive timesteps ($i_j = i_{j+1} \pm 1$) or are only separated by gates which are not single-qubit measurements on w in intervening timesteps $i_j < i < i_{j+1}$.*
4. *every gate $E^{(i_j)}$ encountered in the sequence satisfies the following two conditions:*
   (a) *it is entangling if the path exits it in the forward direction ($i_{j+1} = i_j + 1$)*
   (b) *it is disentangling if the path exits it in the backward direction ($i_{j+1} = i_j - 1$).*

Definition 7. ***Reachability.*** *A qubit u at timestep i is* reachable *from another qubit v in another (possibly the same) timestep i' if there is some path of entangling gates that connects them.*

We now define a standard form for circuit in which circuit coherence will be well-defined.

Definition 8. ***Standard form circuits for calculating circuit coherence.*** *Standard form circuits must have the following properties:*

output qubits $O \subseteq V$**:** *These qubits are semantically defined as containing the useful outputs of a quantum circuit. They do not have to be projectively measured. They may, for example, be the control for a later coherent measurement when cascaded with another quantum circuit.*

input qubits $I \subseteq V$**:** *These qubits are prepared in a classical product state (the computational basis) and are all reachable in timestep* 1 *from the output qubits in timestep* D*.*

ancillae qubits: *these are prepared in the product state of all* $|0\rangle$*'s.*

.

We will assume all circuits from now on are in standard form.

Definition 9. ***Computational subset, computational set, computational state.*** *The computational subset in timestep i (abbreviated* L_i*) is the subset of the qubits which are reachable from the output qubits* O*. The* computational set *is the set of computational subsets across all timesteps.*

$$M = \{L_1, L_2, \ldots, L_D\} \tag{4.9}$$

The computational set are those qubits on which an entangled, coherent quantum state (which we will call the computational state) evolves over time from the initial preparation of the input qubits $I \subset V$ in timestep 1 until the output qubits $O \subset V$ are measured in timestep D. It is measured in qubits, and potentially grows (if entangling gates are applied), shrinks (if disentangling gates including measurements are applied) or stays the same in size in every timestep. We note the following relationships for well-formed circuits.

$$L_1 = I \qquad L_D = O \qquad L_i \subseteq V \tag{4.10}$$

The computational subset can be computed in two passes through the quantum circuit, one forward to determine reachability and one backward to determine the computational subset. This in general requires classical simulation of a quantum circuit, but we can perform this algorithm efficiently on a layered quantum circuit as defined in Section 4.3.1.

In each timestep i, we partition all W qubits in V into disjoint subsets which each contain a self-entangled state. We denote these other (possibly non-computational) qubit subsets as $\{\tilde{L}_i^{(j)}\}$, of which one is the same as the current computational subset L_i.

This partitioning, like L_i, is updated in every timestep.

Following the definition of 2D CCNTCM, each qubit subset is a contiguous subgraph of the main graph G. No two qubit subgraphs share any vertices, but all vertices are covered and the subgraphs may share edges. All entangling/disentangling gates $E_{uv}^{(i)}$ that occur during a timestep i are contained in the set G_i.

Qubit subsets may potentially share common qubits and become entangled by an entangling gate in a past timestep $i' < i$ from the current timestep i. We keep track of all qubit subsets at a given timestep i in a collection $M_i = \{\tilde{L}_i^{(j)}\}$.

4.1.3 The Coherence Calculation Algorithm

The following algorithm takes as input a quantum circuit in standard form with graph $G = (V, E)$, gates in groups G_i over the basis of single-qubit and two-qubit gates $U(2) \cup U(4)$ including single-qubit measurements in the Z-basis. Gates in group G_i execute in timestep i. The algorithm returns as output the computational subset at every timestep $\{L_1, L_2, \ldots, L_D\}$.

The data type of qubit subset $\tilde{L}_i^{(j)}$ is the tuple $(\tilde{V}, P)$ where:

- $\tilde{V} \subseteq V$ are the qubits in the subset
- P is a pointer to the parent qubit subset in timestep $i - 1$, initially NULL.

When we assign one qubit subset to another, we assume we are only assigning the qubits part of the tuple, $\tilde{V}$.

1. Initialize the following:

   - $L_1 \leftarrow \{I\}$.
   - $\tilde{L}_1^{(j)} = v_j \in V \setminus I$ (all non-input qubits v_j get their own subset)
   - $M_1 = \{L_1\} \cup \{L_1^{(j)}\}$.

2. In timestep $i \in \{2, \ldots, D\}$:

   (a) Compute the classical description of the state on all the qubits ρ_i^V from the state ρ_{i-1}^V given M_{i-1}. This is assumed to efficient (e.g. for layered quantum circuits).

   (b) Initialize $M_i \leftarrow \{\}$.

   (c) Create two sets:

   - T_e which contains every two-qubit gate $E_{uv} \in G_i$ that is entangling in the forward direction based on their state ρ^{uv} in timestep i
   - T_d which contains every two-qubit gate $E_{uv} \in G_i$ that is disentangling in the forward direction based on their state ρ^{uv} in timestep i, along with all single-qubit measurements $M_u \in G_i$.

   (d) For every qubit subset $\tilde{L}_{i-1}^{(j)} \in M_{i-1}$:

   i. **Entangling Case.** Check whether $\tilde{L}_{i-1}^{(j)}$ contains a qubit acted upon by a $E_{uv} \in T_e$.

   A. If it does, call that qubit $u \in \tilde{L}_{i-1}^{(j)}$. Check whether v is in any other qubit subset (call it $\tilde{L}_{i-1}^{(j')}$).

   B. If it is, create a new qubit subset $\tilde{L}_i^{(j)}$ equal to the union of the two qubit subsets from step $i-1$:

$$\tilde{L}_i^{(j)} \leftarrow \tilde{L}_{i-1}^{(j)} \cup \tilde{L}_{i-1}^{(j')}$$

   Update the parent pointers of $\tilde{L}_i^{(j)}$ accordingly.

C. Add the current qubit subset to the current timestep's set of qubit subsets M_i.

$$M_i \leftarrow M_i \cup \{\tilde{L}_i^{(j)}\}$$

ii. **Disentangling Case.** Check whether $\tilde{L}_{i-1}^{(j)}$ matches the following two cases and take the corresponding actions.

A. If $\tilde{L}_{i-1}^{(j)}$ contains two qubits u and v acted upon by some $E_{uv} \in T_d$, then check whether E_{uv} is disentangling between any partitioning of $\tilde{L}_{i-1}^{(j)}$ into two subsets $V_1 \ni u$ and $V_2 \ni v$.

B. If it does, add these two subsets to our collection M_i. Set their parent pointers to $\tilde{L}_{i-1}^{(j)}$.

$$M_i \leftarrow M_i \cup \{V_1, V_2\}$$

C. Otherwise, just set the current subset $\tilde{L}_i^{(j)} \leftarrow \tilde{L}_{i-1}^{(j)}$ with the appropriate parent pointer, and add it.

$$M_i \leftarrow M_i \cup \{\tilde{L}_i^{(j)}\}$$

D. If $\tilde{L}_{i+1}^{(j)}$ contains a qubit u acted upon by some $M_u \in T_d$, then create two new qubit subsets. One just removes the qubit u from the current qubit subset. The other is a single-qubit subset consisting only of u.

$$\begin{aligned} \tilde{L}_i^{(j)} &\leftarrow \tilde{L}_{i-1}^{(j)} - \{u\} \\ \tilde{L}_i^{(j')} &\leftarrow \{u\} \end{aligned}$$

Set the parent pointer of $\tilde{L}_i^{(j)}$ accordingly. Add these to our collection.

$$M_i \leftarrow M_i \cup \{\tilde{L}_i^{(j)}, \tilde{L}_i^{(j')}\}$$

(e) For every qubit subset $\tilde{L}_{i+1}^{(j)} \in M_{i-1}$ not operated upon by any of the previous steps, copy it unmodified as $\tilde{L}_i^{(j)}$ into M_i with the appropriate parent pointer.

3. Do a backward pass from the outputs to the inputs to discover the computational subset L_i in each timestep.

   (a) Verify that the output qubits exactly correspond to one of the qubit subsets in M_D. Call this L_D.

   (b) Initialize the computational set $M \leftarrow \{L_D\}$.

   (c) Working backwards for timestep i in $(D, D-1, D_2, \ldots, 3, 2)$:

      i. Find the parent(s) of L_i. Create a new set that is the union of them called L_{i-1} and add them to M.

$$M \leftarrow M \cup \{L_{i-1}\} \tag{4.11}$$

   (d) Verify that $L_1 = I$ are exactly the input qubits.

   (e) Output $M = \{L_1, \ldots, L_D\}$. This is the computational (sub)set of qubits.

Definition 10. ***Instantaneous coherence****. Instantaneous coherence Q_i is the size of the computational subset (in qubits) in timestep i. From the algorithm above,*

$$Q_i = |L_i|. \tag{4.12}$$

Definition 11. ***Circuit coherence****. Circuit coherence Q is the sum of the computation subset size (in qubits) over all D timesteps of a quantum circuit's execution. It is measured in qubit-timesteps, which is the amount of error-correcting effort to maintain the coherent state of one logical qubit for one timestep of a circuit.*

$$Q = \sum_{i=1}^{D} Q_i = \sum_{i=1}^{D} |L_i| \tag{4.13}$$

We will say that a circuit has greater coherence than another circuit if its Q has a higher value. This should not be confused with the meaning of coherence as resisting decoherence.

The following relationships hold with other circuit resources.

$$D \leq S \leq Q \leq D \cdot W \tag{4.14}$$

The first inequality holds in the least parallel case, each of S gates is executed in sequence and $S = D$. The second inequality holds in the least coherent case, when of S gates either entangles or disentangles another qubit from the computational subset in every timestep, and there are no identity wires within the circuit. The third inequality holds in the most coherent case, all of W qubits are part of the computational subset for each of D timesteps.

As an example, we can bound the circuit coherence of modular multiplication of $2 \times n$-bit CSE numbers, as described in Section 1.6.3. The overall width is $W = O(n^3)$ and depth is $D = O(\log n)$, so the coherence is upper-bounded by $O(n^3 \log n)$. We will suggest possible ways to reduce this resource in Section 4.3.

4.2 Measurement-Based Quantum Computing Background

We will now discuss a model of quantum computing that is very different from the circuit model, but has already provided us with tools for parallelism in our nearest-neighbor architectures in Chapters 1 and 2. Measurement-based quantum computing (MBQC) is a general model which creates a large entangled-state on a graph of qubits and performs a pattern of measurements [79]. Later measurements may depend on previous outcomes, and so a classical controller is required to make each measurement adaptive in this way. Each measurement reduces the size of our entangled state, and the measurement operation proceeds physically across our lattice of qubits, from inputs to outputs. We will discuss here a restricted form of MBQC called one-way computing that was proved universal by Raussendorf-Brown-Briegel [80] by translation from an arbitrary n-qubit circuit. This model uses only single-qubit measurements on a regular 2D lattice. We will exclusively consider this model and refer to it at MBQC for the rest of this section, since it is sufficient for discussing depth optimization of quantum circuits.

MBQC is very different from the circuit model, which describes unitary evolution by the application of quantum gates to stationary qubits. Quantum circuits start out with a product state and then slowly build up more and more entanglement until it is finally projectively measured at the end. Surprisingly, both the circuit model and MBQC are

equivalent, but have a tight depth separation for quantum algorithms.

In Section 4.2.1, we will review the MBQC model. In Section 4.2.2, we will discuss the work of Broadbent-Kashefi in automated circuit parallelization, which is a compilation-like process for reducing circuit depth at the expense of size and width, another quantum time-space tradeoff. These represent upper bounds on time-space tradeoffs for mapping certain circuit classes to a nearest-neighbor circuit with classical controller. We will compare this to the time-space tradeoff of other re-ordering networks for mapping circuits to a nearest-neighbor architecture.time-space tradeoffs for low-depth factoring. We conclude by comparing and contrasting the circuit coherence of an MBQC pattern with its other circuit resources.

4.2.1 MBQC Background

We follow the exposition of Ref.'s [21, 27] In the MBQC model, quantum computation is represented by three components: a pattern, an entanglement graph of qubits and interactions, and a classical controller.

A *pattern* is a sequence of commands which come in five types:

N_i**:** preparation of a qubit i into the state $|+\rangle$.

E_{ij}**:** entanglement of qubits i and j with the two-qubit gate $\Lambda(Z)$ defined below. Note that this gate is bidirectional, so it does not matter which qubit is the control or the target.

M_i^{α}**:** single qubit measurement on qubit i which projects onto the states $\{|\pm_\alpha\rangle\}$ where

$$|\pm_\alpha\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm e^{i\alpha} |1\rangle)\} \tag{4.15}$$

Associated with every measurement is a signal $s_i \in \mathbb{Z}_2$ which is 0 for outcome $|+_\alpha\rangle$ and 1 for outcome $|-_\alpha\rangle$.

X_i^s**:** a dependent Pauli X correction, which applies X to qubit i if the signal s is 1.

Z_j^t: a dependent Pauli Z correction, which applies Z to qubit i if the signal t is 1.

A pattern is a valid executable sequence which corresponds to well-defined quantum and classical operations if no command depends on outcomes that are not yet measured. Pattern and executed right to left, much like composing the matrices that make up a sequence of gates to be applied to a quantum state (column vector). Certain operations can be parallelized if they occur on disjoint qubits. Furthermore, the operations in the pattern describe the quantum operations above only. Implicitly inserted in between them are classical layers which compute the dependent signals s, which may be the parity (sum modulo 2) of multiple signals: $s = \oplus_i s_i$.

We illustrate this via a simple pattern on two qubits below.

$$X_2^{s_1} M_1^{-\alpha} E_{12} N_2^0 N_1^0 \tag{4.16}$$

In this pattern, both qubits 1 and 2 are initially prepared in the state $|+\rangle$ and then entangled with a $\Lambda(Z)$ gate. Qubit 1 is measured in the basis $|\pm_\alpha\rangle$ and its classical outcome is stored in the signal s_1. Finally, qubit 2 is corrected with a Pauli X based on the outcome of the measurement.

The entanglement graph $G = (V, E)$ defines all the two-qubit entanglement operations (edges in E) between qubits (vertices in V). Furthermore, there are special vertices which represent the input qubits $I \subset V$ and the output qubits $O \subset V$. In the one-way MBQC model that we are exclusively considering, the geometry of the entanglement graph always has the following form corresponding to an n-qubit circuit: it is a rectangular, regular 2D lattice of $n \times D(n)$ qubits where the leftmost column of n qubits are the inputs and the rightmost column of n qubits are the outputs. We state without proof that an MBQC pattern can be standardized so that measurements and corrections always proceed from left-to-right across this graph.

The preparation commands are often omitted since it is implied that they are always done for all qubits except the input. Measurements can also be done in a basis which depends on previous measurement outcomes. These are written as measurements which

are preceded by some X and Z correction, which themselves are dependent.

$$ {}_t[M_i^\alpha]^s \equiv M_i^\alpha X_i^s Z_i^t = M_i^{(-1)^s \alpha + t\pi} \tag{4.17} $$

An MBQC pattern, since it has circuit-like properties, also consumes circuit depth, width, and size. The depth is divided up into preparation depth (which involves applying the quantum operations N_i and E_{ij}) and computation depth (which involves applying the operations M_i^α, X_i^s, Z_i^t).

A pattern can be optimized in polynomial classical time so that all preparation and entanglement occurs first in the preparation depth, all measurements and X corrections come next in interleaved layers of quantum and classical processing (the computation depth), and finally all the Z corrections come last. The preparation depth is equal to the maximum degree of the underlying graph G, which is always 4. The Z corrections can always be performed last. Therefore our depth bottleneck comes from our measurement commands and their dependencies.

Translations between MBQC patterns and quantum circuits are most easily done using the following (universal) basis of $\{J(\alpha), \Lambda(Z)\}$:

$$ J(\alpha) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & e^{i\alpha} \\ 1 & -e^{i\alpha} \end{bmatrix} \qquad \Lambda(Z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{4.18} $$

Note that for the special angle of 0, $J(0) = H$, our usual Hadamard gate. This is universal because it contains at least one entangling two-qubit gate, and arbitrary single qubit rotations can be implemented using the $\{J(\alpha)\}$ basis as shown below for angles ϕ, β, γ, and δ.

$$ U = e^{i\phi} J(0) J(\beta) J(\gamma) J(\delta) \tag{4.19} $$

However, note that this basis is not fixed and finite. Further restrictions must be placed on MBQC patterns in order to meet fault-tolerance requirements. Namely, the angles α

should be drawn from a finite set that that can be compiled from a fault-tolerant basis such as Clifford+T or Clifford+$Toffoli$, as done by the quantum compilers in Chapter 3.

In fact, we conjecture that because of this fundamental quantum compiling limitation, no MBQC pattern can ever have depth smaller than the corresponding quantum circuit over a fixed, finite basis. This does, however, open up the question of efficient quantum compiling over the $\{\Lambda(Z), J(\alpha)\}$ basis.

4.2.2 *Automating Circuit Parallelism with MBQC*

The work by Broadbent-Kashefi introduced the application of a *measurement calculus* to transform MBQC patterns and provide a pattern for automated parallelization of quantum circuits. This is a classical, compilation-like procedure which takes as input a quantum circuit and returns as output a new quantum circuit with at least the same depth and in some cases improved depth, with a corresponding increase in circuit size and width.

The basic technique involves translating a unitary circuit C from a certain basis into an MBQC pattern P. Two optimizations are used from the measurement calculus: standardization and signal-shifting. Standardization applies the rules below to make sure all patterns are well-formed: all preparation commands precede all entanglement operations, which themselves precede all measurements and corrections. This is useful for standardizing two patterns $P^(1)$ and $P^(2)$, which themselves may be standardized and which are concatenated together. Such a concatenation may occur when we are translating two concatenated unitary circuits which may individually be easy to translate to patterns but together may be difficult. This example is illustrated below, where $C^{(x)}$, $M^{(x)}$, and $E^{(x)}$ correspond to correction operations, measurement operations, and entanglement operations for pattern $P^{(x)}$. The symbol $\rightarrow^*$ indicates the transformation of standardization.

$$P^{(1)} = C^{(1)}M^{(1)}E^{(1)} \tag{4.20}$$

$$P^{(2)} = C^{(2)}M^{(2)}E^{(2)} \tag{4.21}$$

$$P^{(1)}P^{(2)} = C^{(1)}M^{(1)}E^{(1)}C^{(2)}M^{(2)}E^{(2)} \tag{4.22}$$

$$P^{(1)}P^{(2)} \rightarrow^{*} C^{(1)}C^{(2)}M^{(1)}M^{(2)}E^{(1)}E^{(2)} \tag{4.23}$$

$$\tag{4.24}$$

Signal-shifting further optimizes a pattern by moving all Z corrections to the end of the pattern, where they can all be performed in parallel. On a cluster-state graph, the preparation depth and Z-correction depth are then constant. The computation depth of the pattern is dominated by the depth of dependent measurements and X corrections.

Because the cluster state graph is a 2D CCNTC lattice, it is natural to wonder whether the circuit translation techniques of Broadbent-Kashefi can be used to automatically map any quantum circuit to a 2D CCNTC architecture. Indeed, the discovery of a constant-depth teleportation circuit on 2D CCNTC proceeds directly from an MBQC pattern for long-range teleportation. This is illustrated in the next equation, which is a pattern for teleporting qubit i to qubit k.

$$X_k^{s_j} X_k^{s_i} M_j^0 M_i^0 E_{jk} E_{ij} \tag{4.25}$$

Furthermore, the quintessential example of a tight logarithmic separation between the MBQC and circuit model is the parity function. On a non-adaptive quantum circuit (one without a classical controller), the depth of computing parity is $\Omega(\log n)$ [34]. However, as an MBQC pattern, it takes constant depth [21]. Indeed the pattern for parity is very similar to our circuit for unbounded quantum fanout, which is not surprising given that the two functions are related by conjugation of Hadamards on every qubit [73].

Lemma 7.5 from [21] answers affirmatively that any non-nearest-neighbor circuit can be mapped to a nearest-neighbor circuit with constant-depth overhead and the following time-space tradeoff. We restate it here.

Proposition 1. ***Time-space Tradeoff for Mapping Circuits to Nearest-Neighbor*** *[21] Let C be a quantum circuit with depth D, size S, and width W, with J_c the number of $J(\alpha)$ gates, and m the number of places where two $\Lambda(Z)$ operate consecutively on the same qubit. Then the corresponding MBQC pattern P on a cluster-state graph, with the teleportation pattern above, has depth $D' = O(D)$ and width $W' = W + J_c + m = O(W + S)$.*

This gives a time-space (depth-width) tradeoff for compiled, nearest-neighbor circuits of $D'W' = O(D(W+S))$, where D, S, and W are the depth, size, and width of the original, non-nearest-neighbor circuit.

Other than a nearest-neighbor mapping, however, automated techniques cannot provide an asymptotically lower depth for generic circuits. There are other special classes of circuits, namely those composed entirely of Clifford gates and an initial layer of $J(\alpha)$ gates, which can be parallelized to $O(1)$ depth.

We now compare MBQC to two similar automated mappings, or qubit re-ordering networks, for any n-qubit circuit (on AC). The main application of such a re-ordering is to convert quantum circuits to a nearest-neighbor architecture. The re-ordering network of Rosenbaum [82] has the same constant-depth overhead of $D' = O(D)$ but a quadratic width overhead of $W' = O(W^2)$. The re-ordering network of Beals et al. [6] allows a distributed quantum computer with nodes connected in a hypergraph topology (equivalent to 2D CCNTCM) to execute the same circuit with $D' = O(D \log^2 n \log\log n)$.

An MBQC pattern, when executed on a cluster-state graph, has a well-defined circuit coherence based on Section 4.1. For a cluster-state graph with $n \times D$ qubits, including the input and output qubits, corresponding to a unitary circuit on n-qubits and depth D, the circuit width is $W = nD$. The entire lattice starts out in an entangled state, and measurements proceed column-by-column left-to-right from the input qubits to the output qubits. Therefore, the circuit coherence is $Q = D \cdot W = O(n^2 D^2)$ whereas $S = O(nD)$, and in fact, there is no asymptotic separation between circuit coherence and the depth-width product.

4.3 Circuit Coherence as a Time-Space Tradeoff

Although circuit coherence's motivation was to capture another resource of interest to optimize, any new resource may introduce tradeoffs with existing resources. In this case, decreasing circuit coherence may increase size and indirectly depth.

First, we will describe a special form for quantum circuits called layered form in Section 4.3.1. This will make it easier to use time-space tradeoff results from Bennett's reversible pebble game, described in Section 4.3.2. Then in Section 4.4.2, we describe a conjectured asymptotic separation between circuit coherence and the depth-width product for modular multiplication in Shor's factoring algorithm. Finally, we conclude with promising directions for future research regarding factoring architectures and circuit coherence.

4.3.1 Circuits in Layered Form

We now present a special form for quantum circuits which will be useful later in proving facts about circuit coherence, called *layered form*. It applies to quantum algorithms in which gates execute monotonically from input qubits to output qubits in parallel layers, and therefore the physical layout of the qubits mimic the logical execution of gates. This is useful for circuits which leave garbage ancillae behind in each layer. In those cases, circuit coherence can be improved at a negligible increase in circuit depth and size. This is exactly the form of our nearest-neighbor factoring architectures in earlier chapters.

Layered form is later used to leverage results from reversible pebble games on a linear graph. It is also similar to the circuits which can be parallelized to constant-depth MBQC patterns given in Section 4.2.2, except that each layer is allowed to introduce ancillae qubits.

Definition 12. ***Classical layered form for quantum circuits.*** *We say an n-qubit quantum circuit is in* layered form *if the following properties apply. We assume that the circuit is part of an architecture with a classical controller (CCAC).*

1. *All gates are single-qubit or two-qubit gates.*

2. *All qubits can be arranged in a directed, acyclic graph in $\tilde{D}+1$ layers $(l_0, l_1, l_2, \ldots, l_{\tilde{D}})$ such that all two-qubit gates only operate between consecutive layers or within a current layer. The size of a layer is the number of qubits within it, which is polynomial in n: $|l_i| \in poly(n)$.*

3. *The n input qubits are in l_0 and the n output qubits are in $l_{\tilde{D}}$.*

4. *All gates can be partitioned into $\tilde{D}$ cohorts $(C_1, \ldots, C_{\tilde{D}})$, such that all gates in a cohort C_j operate on the same consecutive layers (l_j, l_{j+1}) or within those two layers. This is not a unique partitioning.*

5. *All gates in cohort C_j execute before all gates in cohort C_{j+1}.*

The layers $(l_1, \ldots, l_{\tilde{D}})$ can be considered disjoint groupings in physical space. The gate groups $\{G_j\}$ execute in disjoint cohorts which can be considered groupings in time. At any one timestep, only gates from one cohort are executing, and each cohort executes in order from C_1 to $C_{\tilde{D}}$.

We call the *layer depth* of our layered circuit $\tilde{D}$, the number of physical layers and also the number of cohorts. It is at least the total circuit depth D, with saturation when each cohort only contains gates which execute in a single timestep.

$$\tilde{D} \leq D \tag{4.26}$$

Layered quantum circuits as described above never uncompute a layer. In the worst case, garbage is left behind in every layer until all layers are potentially full of garbage, and we are left with our answer in the output qubits and our input qubits as before.

Most importantly for our proofs below, we can now define *layer coherence* $\tilde{Q}_i$ for timestep i as the number of layers which intersect with the computational subset L_i from our algorithm in Section 4.1.3. The total layer coherence for a circuit is the sum of the layer coherences in any timestep.

$$\tilde{Q} = \sum_{i=1}^{D} \tilde{Q}_i \leq \tilde{D}^2 \tag{4.27}$$

Without uncomputation, each $\tilde{Q}_i$ is equal to i and upper-bounded by $\tilde{D}$, so the total layer coherence $\tilde{Q}$ is upper-bounded by the layer depth squared.

We can define the maximum number of qubits in any layer as w_{max}.

$$w_{max} = \max_{i \in [D]} |l_i| \tag{4.28}$$

Then the total circuit coherence is upper-bounded by the layer coherence multiplied by w_{max}, although this will asymptotically be the same as the depth-width product $D \cdot W$.

$$Q \leq \sum_{i=1}^{D} \tilde{Q}_i \cdot w_{max} = O(D \cdot W) \tag{4.29}$$

We can achieve tighter upper bounds by using the exact layer widths for every layer l_j which intersects with L_i in every timestep i. The inequality below is saturated for $D = 1$.

$$Q \leq \sum_{i=1}^{D} \sum_{j : l_j \cap L_i} |l_j| \leq D \cdot W \tag{4.30}$$

We can now verify that our factoring implementations presented in earlier chapters are in fact layered circuits with the following layer and cohort partitionings. For modular multiple addition, each CSA layer is a cohort which operates in constant depth, therefore, $\tilde{D} = O(D)$. The results of each CSA layer are propagated to the next CSA layer, where in the next cohort, those numbers are also added in constant-depth, and so on. For the partial product creation stage of modular multiplication, we can define that entire stage as a layer for purposes of computing and uncomputing garbage bits, even though it has depth $O(\log n)$, size $O(n \log^2 n)$, and width $O(n^3)$.

So far, we have described a circuit which proceeds through CSA layers (or just layers, to use the more general layered circuit term) to compute a final answer, leaving behind garbage in each layer and never uncomputing them. There is a single "front edge" of computation which proceeds from input qubits to output qubits, both physically over the circuit in layers and temporally over cohorts.

However, it is natural to ask whether we can perform any other kind of intermediate uncomputation to reduce circuit coherence, at the possible increase of either circuit depth or circuit size. The answer to this question is the subject of the rest of this chapter.

4.3.2 *The Pebble Game and Reversible Time-Space Tradeoffs*

An important time-space tradeoff for classical reversible Turing machines originates from the pebble game as studied by Bennett [11]. This is relevant to quantum time-space tradeoffs when executing completely classical circuits on quantum inputs, as in many arithmetic functions and a large part of Shor's factoring algorithm. Moreover, this pebble game models how a reversible machine can compute an irreversible function. It has a direct connection to circuit coherence as we shall see below, since quantum computations, especially low-depth ones, can leave garbage behind which must be uncomputed.

The pebble game is a stylized setting for studying time-space tradeoffs. Although it may take place on general graphs, we study a line graph in analogy to the mechanism of an MBQC pattern and our factoring architectures from Chapters 1 and 2. In short, imagine a row of $n+1$ tiles $(t_0, t_1, \ldots, t_n)$ in sequence, each of which may contain at most one pebble. One pebble is placed on t_0 in the first timestep, and the goal is to place a pebble on tile t_n. The only allowable move is that at every timestep, you may add or remove a pebble from tile $i+1$ if there is a pebble on t_i. Therefore, you can never remove the pebble from tile t_0. (This is known as an input-saving pebble game. There are other versions of the pebble game, similar to other kinds of reversible computations, where the input can be removed).

In what follows, we modify the standard computer science integer set notation to include 0:

$$[T] = \{0, 1, 2, \ldots, T\} \tag{4.31}$$

The number of timesteps it takes to place a pebble on tile n is defined as the time T. The number of pebbles present on all tiles at any one move i is known as the instantaneous space S_i. We call the space S for the whole pebble game as the maximum number of any pebbles on a game at any particular time: $S = \max_{i \in [T]} S_i$. The obvious strategy for winning the pebble game is to place a pebble on t_i in timestep i, without removing any of them. This completes in time $T = n$ and space $S = n$. In the case of unbounded space (unlimited pebbles), this is the optimal depth. However, by bounding space, we can introduce a time-space product TS and attempt to upper-bound and lower-bound it. The

product TS of this naive strategy is n^2.

Knill gave a lower bound for the minimum pebble-game time-space tradeoff [57] which is bounded above by n^2, thereby showing the non-optimality of the naive strategy.

$$TS(n) = 2^{2\sqrt{\log(n)}(1+o(1))}n = o(n^2), \omega(n) \tag{4.32}$$

As a consequence, he obtains a minimum time-space tradeoff for Shor's factoring algorithm on AC.

$$TS(n) = 2^{2\sqrt{n}(1+o(1))}n^3 = o(n^4), \omega(n^3) \tag{4.33}$$

The minimum time-space tradeoff for factoring is indeed consistent with the depth-width product of all known factoring implementations from Table 1.2, including the current work. The one exception is the approximate 1D NTC factoring implementation by Kutin [59], which suggests that a different TS tradeoff may apply for approximate modular exponentiation. This also indicates that the earlier 1D NTC work by Fowler-Devitt-Hollenberg [37] may come the closest to the optimal depth-width product (and therefore optimal circuit coherence) for exact modular exponentiation.

4.3.3 Pebble Games and Layered Quantum Circuits

Taking a step back, how do we determine T? We must return to the original motivation for the pebble game, in simulating an irreversible Turing machine on a reversible one [12]. The irreversible Turing machine's running time on a particular input is defined as T and the maximum space it uses over this time (including the input) is S. From this definition, we get $S \geq T$. What is the corresponding pebble game for the most naive reversible simulation strategy possible?

Definition 13. ***Irreversible pebble game.*** *An irreversible pebble game is one which never removes any pebbles. It corresponds to a computation on an irreversible Turing machine M. By definition, it has time $T(n) = n$ and some space $S(n)$. There is only one irreversible pebble game on n tiles, and it is also called the naive pebbling strategy. In each move i one can only place a*

pebble in tile t_i. We can define the number of pebbles after any move i as the instantaneous space *S_i, which in this case is just i, where $S = S_T$. Therefore, $S = n$.*

Definition 14. ***Pebble-instances.*** *We now define a new quantity* pebble-instances *denoted by $\hat{S}$ which is the total number of pebbles which appear in any history of the pebble game. This is just the sum of all the instantaneous spaces S_i defined above.*

$$\hat{S} = \sum{}_i = 1^T S_i \tag{4.34}$$

We also define the indicator random variable $t_{i,j}$ which is 1 if a pebble is on tile t_i at timestep j and zero otherwise. We can define the instantaneous space in terms of these indicators.

$$S_i = \sum_{j=1}^{T} t_{i,j} \tag{4.35}$$

We can now provide analogous constructions for both an irreversible pebble game and a layered quantum circuit, where the parameters in the pebble game upper bound those in the layered quantum circuit.

Proposition 2. ***Irreversible pebble game for layered quantum circuit.*** *Consider a layered quantum circuit C as defined in Definition 12. Then a pebble game P can be constructed such that time T and space S are equal to layer depth $\tilde{D}$, and the number of pebbles at any timestep i, called S_i, equals the layer coherence $\tilde{Q}_i$.*

Proof. For the given layered quantum circuit C, construct a pebble game P with $\tilde{D}$ tiles. For every cohort execution of C_{i-1}, $i \in [\tilde{D}]$, place a pebble on tile t_i. In the first timestep $i = 0$, note that setting the quantum inputs for C is the same as executing cohort C_0 in layer l_0 and placing a pebble on tile t_0.

Then if there is a pebble on tile t_j, the layer l_j is part of the computation state. In every timestep i then, the layer coherence $\tilde{Q}_i = i$ and the number of pebbles on the tiles is i. Both C and P complete in timestep $\tilde{D}$. □

During Bennett's reversible simulation with time T' and S', there is some overhead. That is, $T' \geq T$ and $S' \geq S$. Bennett's upper bounds for these figures as well as Sherman-Levine's improvements [62] are shown in Table 4.1. We will make use of these constructions in the next section.

Tradeoff Construction	T'	S'
Naive	T	ST
Bennett [12]	$O(T^{1+\epsilon})$	$O(\epsilon 2^{1/\epsilon} S \log T)$
Levine-Sherman [62]	$O(T^{1+\epsilon}/S^{\epsilon})$	$O(\epsilon 2^{1/\epsilon} S(1+\log \frac{T}{S}))$

Table 4.1: Pebble game time-space tradeoffs for reversible simulation of irreversible Turing machines.

The pebble game as we have presented it can be called a serial, reversible pebble game because only one pebble is added or removed at a time. This simplifies our analysis in what follows, but one can also define a parallel pebble game as one where multiple moves (exactly as defined for the serial game above) can be made. The locations where these moves can be made are the boundaries between a pebble on t_i and no pebble on t_{i+1}. We will not discuss this variation any further, but it is a interesting direction for future research to decrease both depth and circuit coherence further.

4.3.4 Instantaneous Coherence and Pebble Game Space

Finally, we conclude by describing the connection between the pebble game and circuit coherence as defined in Section 4.1. To do this, we now construct a reversible pebble game for a modified layered quantum circuit which now allows uncomputation.

We will then use the construction of Bennett [12], which simulates an irreversible Turing machine computation on a reversible Turing machine via a reversible pebble game. We will then use to show how to execute a layered quantum circuit with reduced circuit coherence by completing with only the input and output layers in the computation state.

Definition 15. ***Layered quantum circuit with uncomputation.*** *A layered quantum circuit can be augmented to allow layers to be uncomputed by allowing in timestep i that the cohort C_i can operate as follows:*

- *C_i can operate on any consecutive layers (l_j, l_{j+1}), for any $j \in [\tilde{D}]$, not just layers (l_i, l_{i+1}).*

- *All C_i that operate on a particular layer pair (l_j, l_{j+1}) execute exactly the same gates in the same order, just shifted in time. Therefore, given a computation state in layer l_j, any such C_i either extends the computation state into l_{j+1} if it was previously all $|0\rangle$, or it uncomputes l_{j+1} completely back to all $|0\rangle$ if that layer was already in the computation state.*

Now, we will show that a reversible pebble game can upper-bound the layer coherence of a layered quantum circuit with uncomputation.

Theorem 8. ***Circuit coherence and pebble game space.*** *Given a layered quantum circuit C' with uncomputation with layer depth D and width W, we can construct a one-dimensional reversible pebble game P' that has time T and space S, which executes in parallel timesteps. Then the layer coherence $\tilde{Q}$ is upper-bounded by the total pebble-timesteps $\mathcal{S}$, and the total circuit coherence is upper-bounded by the following weighted sum:*

$$Q \leq \sum_{i=1}^{D} \sum_{j=1}^{D} t_{i,j} |l_j| \tag{4.36}$$

Proof. This construction mirrors the one in Theorem 2, except now in timestep i, when the cohort C_i in C' uncomputes a layer l_j from the layer l_{j-1}, in P' we remove the pebble from tile t_j and there is a pebble on tile t_{j-1}. Therefore, we maintain the following invariants at every timestep, starting with $i = 1$.

$$S_i = \tilde{Q}_i \tag{4.37}$$

$$\hat{S} = \tilde{Q} \tag{4.38}$$

Now, unlike in the pebble game, where all pebbles on any tile t_i represent unit space, the corresponding circuit layer l_i to that tile represents $|l_i|$ qubits as part of the computation state. Therefore, the instantaneous coherence Q_i at timestep i is equal to the weighted sum of the pebbles which are present in the same timestep.

$$Q_i = \sum_{j=1}^{D} t_{i,j} |l_j| \tag{4.39}$$

Since the total circuit coherence is the sum of the instantaneous coherences, we have the desired result.

$$Q = \sum_{i=1}^{D} Q_i = \sum_{i,j=1}^{D} t_{i,j}|l_j| \tag{4.40}$$

□

4.4 Circuit Coherence of Factoring Architectures

In this section, we apply circuit coherence and pebble-game uncomputing techniques from the previous section to our factoring architectures. The techniques are generic for any layered circuit, and are not specific to factoring or even nearest-neighbor circuits, except that the layered circuits are nearest-neighbor at the layer level.

In Section 4.4.1, we provide a conjecture for decreasing circuit coherence for modular multiplication while only marginally increasing our depth.

In Section 4.4.2, we provide a generalized, configurable-depth factoring architecture based on Chapter 1.

4.4.1 Reducing Circuit Coherence with Intermediate Uncomputing

The naive strategy for computing modular multiplication has depth $D = O(\log n)$, size $O(n^2)$, and width $O(n^3)$. Therefore, the circuit coherence is upper-bounded by $O(n^3 \log n)$. However, Bennett [12] with corrections from Sherman-Levine [62] have shown that an irreversible pebble game with time T and space S can be simulated reversibly with overhead $T' = O(T^{1+\epsilon})$ and $S = O(\epsilon 2^{1/\epsilon} S \log T)$ Therefore, we propose the following conjecture.

Conjecture 3. *Define a layered circuit C for modular multiplication with layer depth $\tilde{D} = O(\log n)$ and maximum layer width $w_{max} = O(n^3)$. Define P as the pebble game corresponding to the optimal reversible simulation of an irreversible pebble game on $\tilde{D}$ tiles on a reversible pebble game with the same number of tiles, as proved by Li [63] based on the authors above. Using the results of Theorem 8, we can create a new layered circuit C' with uncomputation that performs the same modular multiplication with the following conjectured resources: depth $D' = O(D^{1+\epsilon})$, the same width $W' = W$ and reduced circuit coherence $Q = O(\epsilon 2^{1/\epsilon} n^3 \log\log n)$.*

4.4.2 *Configurable-Depth Factoring*

When we decrease our nearest-neighbor factoring depth from $O(\log^2 n)$ in Chapter 1 to $O((\log\log n)^2)$ in Chapter 2, we calculated a disproportionate increase in circuit size and width from $O(n^4)$ to $\Omega(n^6 \log^2 n)$. This seems to be quite an unfavorable depth-width tradeoff, and it is natural to ask whether we could have some configurable depth in between polylogarithmic and sublogarithmic that would let a quantum systems architect choose the right tradeoff for a particular implementation.

In this section, we provide such a configurable depth factoring architecture by generalizing our implementation in Chapter 1. In that chapter, we wanted to multiply $n \times n$-bit quantum integers in parallel. To do so, we divided them up into $\lceil n/2 \rceil$ groups of two. In each group of two quantum integers, call them $|x\rangle$ and $|y\rangle$, in order to get all the product bits $|x_i \cdot y_j\rangle$ we need to generate $n^2 \times n$-bit modular residues $2^i 2^j \bmod m$ controlled on two qubits. We then add these down with modular multiple addition back to an n-bit (CSE) number, and we are then left with $\lceil n/2 \rceil$ numbers to multiply in the second level. This takes place in depth $O(\log n)$ and takes width and size $O(n^3)$. Modular exponentiation has $\lceil \log_2 n \rceil + 1$ such levels and perform $(n-1)$ multiplications total, giving us the final depth of $O(\log n)$ and size and width $O(n^4)$.

The configurable parameter is how we group quantum integers for expansion. If instead of groups of two, we used groups of 4, then each multiplication would require expansion into $n^4 \times n$-bit numbers. These would still add down to a single $O(n)$-bit CSE number in $O(\log n)$ depth, but would now take size and width $O(n^5)$. Modular exponentiation would still take total depth $O(\log^2 n)$ but would now take size and width $O(n^6)$, with hidden constants dependent on the parameter.

We now name the configurable parameter d, where in each level of modular exponentiation we group quantum integers into groups of 2^d, and we also expand into $n^{2^d} \times n$ product bits. We compare this new depth-width tradeoff in Table 4.2.

The depth-width product for configurable parameter d is $DW = O(n^{2^d+3})$. By setting it to be less than one $d = 1/k$ for $k \geq 2$, we are actually splitting each n-bit number into 2^{k-1} pieces and grouping them together.

Implementation	D	W
Polylog Depth ($d = 1$), Chapter 1	$O(\log^2 n)$	$O(n^4)$
Config Log Depth (d)	$O(\frac{2^d}{d} \log^2 n)$	$O(\frac{1}{2^d} n^{2^d+2})$
($d = \lceil \log_2 n \rceil$)	$O(n \log n)$	$O(n^{n+2})$
($d = 1/n$)	$O(n 2^{1/n} \log^2 n)$	$O(\frac{1}{2^{1/n}} n^{2^{1/n}+2})$
Sublog Depth, Chapter 1	$O((\log \log n)^2)$	$O(\frac{1}{\epsilon} n^{6+2\epsilon} \log^2 n)$

Table 4.2: A comparison of configurable-depth factoring architectures and their depth-width tradeoffs.

Finally, we suggest a construction for low-coherence factoring circuits. In the original quantum period-finding (QPF) procedure of Nielsen-Chuang [76], n quantum integers are multiplied in series, giving a depth in multiplications of $O(n)$. In the parallelized construction of Kitaev-Shen-Vyalyi [53], all n quantum integers are multiplied in parallel with a multiplication depth of $O(\log n)$. Instead, one can consider another kind of configurable-depth QPF where $n/2^{d-1}$ integers are multiplied in parallel, and there are 2^{d-1} such groups multiplied in serial, where $d = \lceil \log_2 n \rceil$ for the serial QPF above, and $d = 1$ for the parallel QPF above.

Determining other bounds for circuit coherence and extending this to other quantum algorithms remains a promising area for future research.

Chapter 5
CONCLUSION

Quantum architecture is the intermediate layer between algorithms and hardware. It is the design of how quantum bits and their allowed interactions in order to solve these algorithms efficiently on realistic models of quantum hardware. It aims to minimize circuit resources of interest, namely depth, size, and width. In analogy to classical circuits, the depth is the running time of an algorithm allowing parallelization, the size is the number of operations required over all parallel processors, and the width is the number of (quantum) bits required over all parallel processors.

In this dissertation, we study quantum architecture in the context of optimizing Shor's factoring algorithm on nearest-neighbor architectures with realistic constraints. It is hoped that lessons learned in this special-case can contribute to the general community general principles which can be used to generalize other quantum algorithms. We posit that the larger overall goal of quantum architecture should be the design of a general-purpose quantum processor, one which can execute any quantum algorithm with emphasis on being able to perform a core group of operations efficiently. This core group of operations is defined by the instruction set. Once quantum architecture has progressed to this point, we can leverage the remarkable strides in normal digital architecture over the past 80 years.

As well, we may be able to use the insights of quantum architecture to build quantum processors which can surpass digital architectures in the solution of large-scale problems over exponentially-sized solution spaces.

Appendix A

KSV ERROR FROM EARLY MEASUREMENT

In this note, we provide a detailed calculation for the error in phase estimation due to Kitaev, Shen, and Vyalyi (KSV) where projective measurement is used instead of a coherent measurement. We show an increase from $2\sqrt{\epsilon}$ to $\sqrt{2}\sqrt[4]{\epsilon}$. The use of projective measurement is to offload many trigonometric operations onto a classical controller instead of doing them reversibly on a quantum computer. The cost of projective measurement is leaving garbage qubits in the ancillae of the target register, but this is a constant amount relative to the size of circuits we may wish to compile using the KSV procedure. Our goal is to show that this increase in error by a square root factor is negligible and may be preferable in realistic implementations.

To begin with, we review (coherent) measurement operators as a generalization of controlled quantum operators, and then we further extend them to the case of approximately measuring functions on orthogonal basis decompositions where there are garbage bits left over in an ancillary register which must be uncomputed. Next, we calculate the error of this approximate measurement with ancillae. Then, we show how this general measurement procedure corresponds to estimating the phase of the modular multiplication operator used in Shor's factoring algorithm. Finally, we show that the error only increases by a square root factor when we projectively measure the garbage in the ancillae instead of coherently simulating the measurement.

A.1 Measurement Operators

First we introduce some preliminary definitions related to measurement.

A very common quantum operation entangles the results of one register (called the *target*) based on the value of another register (called the *control*). The most basic case of this is CNOT, or $\Lambda(X)$, operator, which operates on a control register of one qubit and a

target register of one qubit.

$$\Lambda(X)\,|y,z\rangle \rightarrow |y,z\oplus y\rangle \tag{A.1}$$

We have two ways of describing how CNOT is measuring in this case. We can say CNOT is *measuring with respect to the decomposition* of the control registers, namely the computational basis, in that the operation on the target register (flipping the bit) depend on decomposing the control register in the basis $\{0,1\}$. We can also say that CNOT is *measuring a function* $f : \{0,1\} \rightarrow \{0,1\}$ which in this case is simply the copy operator, $f(x) = x$. We can rewrite the equation above as:

$$\Lambda(X)\,|y,z\rangle \rightarrow |y,z\oplus f(y)\rangle \tag{A.2}$$

However, we know that the distinction between control and target registers often depends on a particular basis. For example, we can flip the direction of control and target in the CNOT case by conjugating both qubits with Hadamard operators. The general characteristic of measurement operators is that they are entangling, and that they can encode information about one register in another in a very general way. From the point of view of measurement, we call the control register the *object*, as in the state we are trying to measure, and the target register is the *instrument*, as in the state that we transform according to projecting the measurement object in some orthogonal decomposition.

Let's begin with a simple but more general case of a measuring operator W which operates on a space decomposed into the subsystems $\mathcal{N}$ (the measurement object) and $\mathcal{K}$ (the measurement instrument) according to the orthogonal decomposition $\mathcal{N} = \bigoplus_j \mathcal{L}_j$, so-called because each of the $\mathcal{L}_j$ are pairwise orthogonal subspaces. W will perform a different unitary operator U_j on the subsystem $\mathcal{K}$ depending on the projection of $\mathcal{N}$ into each $\mathcal{L}_j$.

$$W = \sum_{j\in\Omega} \Pi_{\mathcal{L}_j} \otimes U_j \tag{A.3}$$

An interesting fact about this definition of measurement operators is that approximativeness is preserved. If each unitary U_j is replaced with another unitary $\tilde{U}_j$ that

approximates it with precision δ, then the new measuring operator $\tilde{W}$ also approximates the original W with the precision δ.

This precision is defined in terms of the inner product between any state $|\zeta\rangle$ operated on by W and by $\tilde{W}$. We can decompose $|\zeta\rangle$ into two subsystems $|\psi\rangle$ and $|\phi\rangle$ corresponding to the spaces $\mathcal{N}$ and $\mathcal{K}$ above. We will use this fact later.

$$\langle\zeta| W^\dagger \tilde{W} |\zeta\rangle = \langle\psi| \otimes \langle\phi| W^\dagger \tilde{W} |\psi\rangle \otimes |\phi\rangle = \sum_{j\in\Omega} \left(\langle\psi| \Pi_{\mathcal{L}_j} |\psi\rangle \otimes \langle\phi| U_j^\dagger \tilde{U}_j |\phi\rangle \right) \leq \delta \quad \text{(A.4)}$$

A.2 Notation

A random section on notation. I will find a better place to put this later.

In [53], ancillae are denoted as $\mathcal{B}^{\otimes N}$ to mean appending N qubits to a quantum state. However, we will use the more general symbol $\mathcal{J}$ to mean any ancillary register of N qudits of dimension d. In context, when we mean an ancillae state initialized to all zeros, we will write $\left|0^N\right\rangle$ where $N = \log_d(\dim(\mathcal{J}))$.

In these notes we will mainly deal with unitary operators U, which by definition operate on states from some vector space $\mathcal{V}$ without enlarging or shrinking the space. We can think of them as being square $\dim(\mathcal{V}) \times \dim(\mathcal{V})$ unitary matrices. However, mathematically adding ancillae qubits enlarges our Hilbert space (say from $\mathcal{V}$ to $\mathcal{V} \otimes \mathcal{J}$), and discarding ancillae qubits shrinks our Hilbert space (say from $\mathcal{V} \otimes \mathcal{J}$ back to $\mathcal{V}$), and these are more properly represented by isometries $\hat{U}$, or non-square matrices of dimension $\dim(\mathcal{V}) \times \dim(\mathcal{V}) \cdot \dim(J)$ (or the transpose of that), which are unital in that $\hat{U}\hat{U}^\dagger = I$. To convert between the two matrix sizes, so that we can combine unitary and unital operators, we can use the standard embedding as in [53]:

$$\forall |\xi\rangle \in \mathcal{V} \qquad V |\xi\rangle = |\xi\rangle \otimes \left|0^N\right\rangle \quad \text{(A.5)}$$

However, in these notes, we take the equivalent approach of just tensoring a unitary operator without ancillae with identity on the ancillae Hilbert space ($U \otimes I_{\mathcal{B}^{\otimes N}}$).

For unitary operators, the input and output vector spaces are the same, so we will often just write them in the following notation instead of the usual $U : \mathcal{V} \to \mathcal{V}$.

$$U : \mathcal{V} \tag{A.6}$$

A.3 Operators That Measure a Function

We now introduce the idea that an operator can measure a function from the indices of the measurement object space $\mathcal{N}$ to the measurement instrument space $\mathcal{K}$ with respect to some fixed orthogonal decompositions of these spaces.

$$\mathcal{N} = \bigotimes_{j \in \Omega} \qquad \mathcal{K} = \bigotimes_{y \in \Delta} \qquad f : \Omega \to \Delta \tag{A.7}$$

Saying that an operator Y is measuring with respect to a fixed orthogonal decomposition Ω is equivalent to saying that Y measures the function f, which need not even be reversible. The simplest, but not the most general, form of such an operator projects the first subsystem $\mathcal{N}$ into a subspace $\mathcal{L}_j$ and performs the corresponding operation $Q_{f(j)}$ on the second subsystem $\mathcal{K}$. In Equation A.8, we define Y as the sum of these projectors, and our notation means it operates on the space of $\mathcal{N}$ tensored with $\mathcal{K}$.

$$Y = \sum_{j \in \Omega} \Pi_{\mathcal{L}_j} \otimes Q_{f(j)} : \mathcal{N} \otimes \mathcal{K} \tag{A.8}$$

A.4 Measurement Operators with Ancillae

However, this is not the most general case of a measurement operator. We make *three extensions* here which require cascading two rounds of measurement through an ancillary register. We now define a new composite measurement operator $\tilde{Y}$ on this space with three subsystems corresponding to the input register $\mathcal{N}$, the intermediate ancillary register $\mathcal{B}^N$, and a final output register $\mathcal{K}$ which approximates Y from Equation A.8.

$$\tilde{Y} : \mathcal{N} \otimes \mathcal{B}^N \otimes \mathcal{K} \tag{A.9}$$

We will build up to an operational definition of $\tilde{Y}$ later, but first we must motivate what it must achieve.

In the first round, we measure with our object in $\mathcal{N}$ and our instrument in $\mathcal{B}^N$. Here we allow for garbage, which is our first extension, described in A.4.1. In the second round, we measure with our object being the instrument of the first round, in $\mathcal{B}^N$, and our instrument is in $\mathcal{K}$. Here we allow for operations other than just copying from $\mathcal{B}^N$ to $\mathcal{K}$, which is our second extension, described in A.4.2. Finally, instead of implementing our ideal Y directly, we allow ourselves, and quantify what it means, to approximate it as $\tilde{Y}$.

A.4.1 First Extension: Measurement with Garbage

To motivate why we need this new composite measurement Y instead of just using W directly from the previous section, let's consider the problem of garbage. In general, only some of the information computed by W into its target register $\mathcal{K}$ above may be useful, but we generate some garbage qubits (say, to make the operation reversible). To be concrete, let's say that we have the following:

$$W : \mathcal{N} \otimes \mathcal{B}^N = \sum_{j \in \Omega} \Pi_{\mathcal{L}_j} \otimes U_j \qquad U_j : \mathcal{B}^N \to \mathcal{B}^N \qquad U_j \left|0\right\rangle = \sum_{y,z} c_{y,z}(j) \left|y, z\right\rangle \tag{A.10}$$

where $y \in \mathbb{B}^m$ represents the useful part of the result, $z \in \mathbb{B}^{N-m}$ is garbage, and the complex weights $c_{y,z} \in \mathbb{C}$ are functions of the index of the orthogonal decomposition of $\mathcal{N}$, the number j. In many cases, we would like to uncompute this garbage in order to use space efficiently. We can do this using an uncomputing trick due to Charlie Bennett [11] by first running an operation U which may produce garbage, copying out the useful part of the result to a second register, and finally running $U^\dagger$ to uncompute the input register.

Now we will give a more concrete definition for Y in terms of W and the operation V which simply copies a state $\left|\psi\right\rangle \in \mathcal{B}^N$ from the ancillary register to a state $\left|\phi\right\rangle in \mathcal{K}$ in the output register using bitwise CNOTs, or bitwise addition modulo 2^n for an n-qubit register. Then $Y = W^{-1}VW$, where W is our operator from above which computes

a function with garbage, V is our copy operation, and W^{-1} uncomputes the original function and it garbage. We make this more rigorous in Equation A.13.

$$Y = WVW^{-1} : \mathcal{N} \otimes \mathcal{B}^N \otimes \mathcal{K} \tag{A.11}$$

$$W = \sum_{j \in \Omega} \Pi_{\mathcal{L}_j} \otimes U_j \otimes I_{\mathcal{K}} \tag{A.12}$$

$$V : |x\rangle \otimes |y\rangle \otimes |z\rangle \rightarrow |x\rangle \otimes |y\rangle \otimes |z \oplus y\rangle \tag{A.13}$$

A.4.2 Second Extension: Measurement with a Non-Copy Operation

Our notion of measurement with garbage is still not the most general it could be. For example, there is no reason why the decomposition of $\mathcal{K}$ has to be the same as that for $\mathcal{B}^N$ or $\mathcal{N}$, or why we are limited to strictly copying the useful output from $\mathcal{B}^N$ into $\mathcal{K}$. We don't need to limit the function f measured by W to be invertible. Furthermore, for practical reasons, there may be a more efficient way of encoding the useful part of f's output, or we may need to do further processing on it as part of the algorithm we want to run.

To allow this, in this section, we allow an arbitrary orthogonal decomposition $\mathcal{K} = \bigotimes_{y \in \Delta} \mathcal{M}_y$. This is the same as allowing V to be an arbitrary sum of projectors onto $\{\mathcal{M}_y\}$ and corresponding unitary operators Q_y on $\mathcal{K}$.

$$V = \sum_{y \in \Delta} I_{\mathcal{N}} \otimes Q_y \otimes \Pi_{\mathcal{M}_y} \tag{A.14}$$

A.4.3 Third Extension: Approximate Measurement

We can now allow a very general form of measurement which allows garbage, non-copying uncomputation, and finally, approximating a function. We will explain this last feature in this section.

We review that in the last two subsections we have been building up a composite measurement operator $\tilde{Y} = W^{-1}VW$ which consists of these two operations on a space

with three subsystems.

$$W = \sum_{j \in \Omega} \Pi_{\mathcal{L}_j} \otimes R_j \otimes I_{\mathcal{K}} \qquad V = \sum_{y \in \Delta} I_{\mathcal{N}} \otimes \Pi_{\mathcal{M}_y} \otimes Q_y \tag{A.15}$$

We now reveal that $\tilde{Y}$ approximates Y from Equation A.8. What does this mean? For one operator, say $\tilde{X}$, to approximate another, say X, within an error δ means that the minimum overlap between any state $|\eta\rangle$ that is operated on by $\tilde{X}$ and X is at least $1 - \delta$. This is shown in Equation A.16.

$$\langle \eta | \, \tilde{X}^\dagger X \, | \eta \rangle \geq 1 - \delta \tag{A.16}$$

Now how do we relate this to approximating a function f between the orthogonal decomposition indices of $\mathcal{N}$ and $\mathcal{K}$? And what does this latter concept mean? It means that the conditional probabilities of getting a given output index $y \in \Delta$ given an input index $j \in \Omega$, namely $P(y \mid j) = \langle 0^N | \, R_j^\dagger \Pi_{\mathcal{M}_y} R_j \, | 0^N \rangle$ satisfies Equation A.18. Approximating a function f with error probability ϵ means that the probability of getting the correct answer $f(j)$ is bounded below by $1 - \epsilon$, and the probability of getting any $y \neq f(j)$ is bounded above by ϵ.

$$P(f(j) \mid j) \;\; \geq \;\; 1 - \epsilon \tag{A.17}$$

$$\sum_{y \neq f(j)} P(y \mid j) \;\; < \;\; \epsilon \tag{A.18}$$

In the next section, we will delve into detailed calculations of relating δ and ϵ given the framework we have built up until now.

A.5 Error Calculations

In this section, we will calculate two error probabilities δ for an approximate measuring operator $\tilde{Y}$, given that we are able to approximate a function f with error probability ϵ. The first error probability, which follows exactly the development in Problem 12.2 of Ref [53], assumes that we measure *coherently*, meaning that at no point do we projectively

measure, and we are able to perfectly uncompute all garbage. This is the ideal case, which we calculate in A.5.1. The second error probability involves projectively measuring as part of the operator W, which involves some purely classical reversible operations that can be offloaded to a classical controller. Afterwards, we execute V as before, but it is now impossible to run W^{-1} because of the projective measurement. This leaves some amount of garbage in the ancillary subsystem $\mathcal{B}^N$, and we calculate it in A.5.2.

A.5.1 Error Probability With Coherent Measurement

First, we use this preliminary lemma. using properties of the operator norm.

Lemma 5 (Solution 12.1, p. 230 [53])**.** *Let $X_j : \mathcal{N}_j \to \mathcal{M}_j$ be a collection of operators which operate on pairwise orthogonal subspaces, where each X_j takes $\mathcal{N}_j$ to $\mathcal{M}_j$. Then the norm of the operator X formed as a direct product of these X_j's has an operator norm equal to the maximum of any of the X_j's.*

$$X = \bigoplus_j X_j : \bigoplus_j \mathcal{N}_j \to \mathcal{M}_j \tag{A.19}$$

$$||X|| = \max_j ||X_j|| \tag{A.20}$$

Proof. This follows from the fact that the operator norm measures how much an operator scales any non-zero vector. If the vector comes from the space which is the direct sum of the $\mathcal{N}_j$'s, it is in a particular fixed subspace $\mathcal{N}_j$. Therefore, it cannot be scaled more than the maximum operator norm of any of the X_j's. □

We now apply this to the case of unitary operators to show how to approximate a measuring operator with ancillae which is the direct sum of projectors onto pairwise orthogonal subspaces.

Lemma 6 (Problem 12.1 [53])**.** *Let W be a unitary operator which acts on a space with two subsystems $\mathcal{N}$ and $\mathcal{K}$ and is the direct sum of projectors onto pairwise orthogonal subspaces of $\mathcal{N} = \oplus_j \mathcal{L}_j$ tensored with unitary operators on $\mathcal{K}$. Let $\tilde{W}$ be an analogous operator except that the unitaries $\tilde{U}_j$ now operate on $\mathcal{K}$ tensored with an ancillary subsystem $\mathcal{B}^N$.*

$$U_j : \mathcal{K} \tag{A.21}$$

$$\tilde{U}_j : \mathcal{K} \otimes \mathcal{B}^{\otimes N} \tag{A.22}$$

$$W : \mathcal{N} \otimes \mathcal{K} = \bigoplus_j \Pi_{\mathcal{L}_j} \otimes U_j \tag{A.23}$$

$$\tilde{W} : \mathcal{N} \otimes \mathcal{K} \otimes \mathcal{B}^N = \bigoplus_j \Pi_{\mathcal{L}_j} \otimes \tilde{U}_j \tag{A.24}$$

Suppose that for each j, $\tilde{U}_j$ approximates (with ancillae) U_j with error ν according to the definition below.

$$\forall_j ||\tilde{U}_j - (U_j \otimes I_{\mathcal{B}^{\otimes N}})|| \leq \nu \tag{A.25}$$

Then the measuring operator $\tilde{W} = \oplus_j \Pi_{\mathcal{L}_j} \otimes \tilde{U}_j$ approximates with ancillae the measuring operator $W = \sum_j \Pi_{\mathcal{L}_j} \otimes U_j$ with the same error ν.

$$||\tilde{W} - (W \otimes I_{\mathcal{B}^{\otimes N}})|| \leq \nu|| \tag{A.26}$$

Proof. We decompose W using its definition.

$$\begin{aligned} ||\tilde{W} - (W \otimes I_{\mathcal{B}^{\otimes N}})|| &= ||(\bigoplus_j \Pi_{\mathcal{L}_j} \otimes \tilde{U}_j) - (\bigoplus_j \Pi_{\mathcal{L}_j} \otimes U_j \otimes I_{\mathcal{B}^{\otimes N}} && \text{(A.27)} \\ &= ||\bigoplus_j \Pi_{\mathcal{L}_j} \otimes (\tilde{U}_j - (U_j \otimes I_{\mathcal{B}^{\otimes N}}))|| && \text{(A.28)} \\ &\leq \max_j ||\Pi_{\mathcal{L}_j} \otimes (\tilde{U}_j - (U_j \otimes I_{\mathcal{B}^{\otimes N}}))|| && \text{(A.29)} \end{aligned}$$

In the last step above, we use Lemma 6 to reduce the error of approximating W with ancillae to the largest error of approximating any U_j with ancillae. Now we use Equation A.25 and continue.

$$\begin{aligned} ||\tilde{W} - (W \otimes I_{\mathcal{B}^{\otimes N}})|| &\leq \max_j ||(\tilde{U}_j - (U_j \otimes I_{\mathcal{B}^{\otimes N}})|| && \text{(A.30)} \\ &= \nu && \text{(A.31)} \end{aligned}$$

In the step above, we used the fact that when taking the norm of an operator which is a projector onto a subspace tensored with a unitary, this reduces to taking the norm of just the unitary. This is because the vectors outside of the subspace, which are projected away to the zero vector, cannot affect the operator norm.

Therefore, we have that $\tilde{W}$ *with* ancillae approximates $\tilde{W}$ *without* ancillae with the same error ν of approximating the unitaries with ancillae within any subspace. □

We now return to the setting of Section A.4.3 and repeat its definitions here. Given a measuring operator W which approximately measures a function, what is the error of approximation of a two stage measurement using W and intermediate ancillae? We will see later that such an approximate two-stage measurement corresponds to parallelized phase estimation.

Theorem 9 (Problem 12.2, [53]). *We consider the space $\mathcal{N} \otimes \mathcal{B}^{\otimes N} \otimes \mathcal{K}$ with the following orthogonal subsystem decompositions: $\mathcal{N} = \bigoplus_{j\in\Omega} \mathcal{L}_j$ and $\mathcal{B}^{\otimes N} = \bigoplus_{y\in\Delta} \mathcal{M}_y$. We define two operators, W which measures $\mathcal{N}$ as object into $\mathcal{B}^{\otimes N}$ as instrument and V which measures $\mathcal{B}^{\otimes N}$ as object into $\mathcal{K}$ as instrument.*

$$W : \mathcal{N} \otimes \mathcal{K} \tag{A.32}$$

$$\tilde{W} : \mathcal{N} \otimes \mathcal{K} \otimes \mathcal{B}^N \tag{A.33}$$

If W approximately measures a function $f : \Omega \to \Delta$ with error ϵ, then the operator $\tilde{Y} = W^{-1}VW$ approximates the following operator Y with error $2\sqrt{\epsilon}$.

Proof. Even though we have already defined Y and $\tilde{Y}$ in terms of operators on the whole space $\mathcal{N} \otimes \mathcal{K} \otimes \mathcal{J}$, it is now useful to define $\tilde{Y}$ in an alternate way: operators P_j which operate on the space $\mathcal{K} \otimes \mathcal{J}$ which can further be expressed in terms of operators Q_y operating on $\mathcal{K}$ and operators R_j operating on $\mathcal{J}$.

$$\tilde{Y} = \sum_{j\in\Omega} \Pi_{\mathcal{L}_j} \otimes P_j \qquad P_j = \sum_{y\in\Delta} Q_y \otimes (R_j^\dagger \Pi_{\mathcal{M}_y} R_j) \tag{A.34}$$

Using the results of Lemma 6, we only need to show that P_j approximates (using ancillae) $Q_{f(j)}$ (without using ancillae) for all j.

To begin with, let's examine the action of P_j and $Q_{f(j)}$ on an arbitrary state $|\xi\rangle \in \mathcal{K}$, possibly augmented with ancillae $\left|0^N\right\rangle$. We further define the following states and the difference between them.

$$|\eta\rangle = Q_{f(j)} |\xi\rangle \tag{A.35}$$

$$|\tilde{\eta}\rangle = P_j(|\xi\rangle \otimes \left|0^N\right\rangle) \tag{A.36}$$

$$|\psi\rangle = |\tilde{\eta}\rangle - |\eta\rangle \tag{A.37}$$

We want to minimize the norm of $|\psi\rangle$, which represents the error in a state operated on by the desired unitary $Q_{f(j)}$ and the state operated on by the approximation with ancillae P_j.

That's how we begin these next calculations.

$$\langle\psi|\psi\rangle = \left[\langle\tilde{\eta}| - (\langle\eta| \otimes \left\langle 0^N\right|)\right] \left[|\tilde{\eta}\rangle - (|eta\rangle \otimes \left|0^N\right\rangle)\right] \tag{A.38}$$

$$= [\langle\tilde{\eta}|\tilde{\eta}\rangle] - \tag{A.39}$$

$$\left[(\langle\eta| \otimes \left\langle 0^N\right|)(|\tilde{\eta}\rangle)\right] - \tag{A.40}$$

$$\left[(\langle\tilde{\eta}| \otimes (|\eta\rangle \otimes \left|0^N\right\rangle)\right] + \tag{A.41}$$

$$[\langle\eta|\eta\rangle] \tag{A.42}$$

$$= 2 - \left(\langle\eta| \otimes \left\langle 0^N\right|\right) |\tilde{\eta}\rangle - \tag{A.43}$$

$$\langle\tilde{eta}| \left(|\eta\rangle \otimes \left|0^N\right\rangle\right) \tag{A.44}$$

In the last line, we use the fact that $|\eta\rangle \otimes \left|0^N\right\rangle$ and $|\tilde{\eta}\rangle$ are both normalized states of unit norm. The two braket terms remaining are complex conjugates of each other (call them $a + bi$ and $a - bi$) so their sum is just $2a$, or the real part of either complex number.

This is where we continue our calculations, using the definition of $|\eta\rangle$ and $|\tilde{\eta}\rangle$ to express their overlaps in terms of Q_y, R_j, and projectors onto the orthogonal decomposition of $\mathcal{J}$.

$$\langle\psi|\psi\rangle = 2 - 2\Re\left[\left(\langle\eta|\otimes\left\langle 0^N\right|\right)|\tilde{\eta}\rangle\right] \tag{A.45}$$

$$= 2 - 2\Re\left[\sum_{y\in\Delta}\langle\xi|\,Q_{f(j)}^{\dagger}Q_y\,|\xi\rangle\left\langle 0^N\right|R_j^{\dagger}\Pi_{\mathcal{M}_y}R_j\left|0^N\right\rangle\right] \tag{A.46}$$

In the last line above, we use the fact that we can distribute a braket through a tensor product. That is, if $|\alpha\rangle$, $|\beta\rangle$, and $|\gamma\rangle$ are normalized vectors, then $(\langle\alpha|\otimes\langle\beta|)\,|\gamma\rangle = \langle\alpha|\gamma\rangle\langle\beta|\gamma\rangle$.

We can now factor out the braket $\left\langle 0^N\right|R_j^{\dagger}\Pi_{\mathcal{M}_y}R_j\left|0^N\right\rangle$ from the product inside the real operation above since the braket is completely real. This is because $\Pi_{\mathcal{M}_y}R_j\left|0^N\right\rangle$ is either the zero vector or a normalized state. We will henceforce call this projected state $\left|\phi_{(y,j)}\right\rangle$, and we will write its overlap with itself as the braket $\langle\phi|\phi\rangle$.

Now we also need to use the fact that the real part of a sum over complex numbers is at most the sum of real parts of each complex number.

$$\Re\left[\sum_i c_i\right] \leq \sum_i \Re(c_i) \tag{A.47}$$

Furthermore, the following inequality holds.

$$1 - \sum_i \Re(c_i) \leq \sum_i (1 - \Re(c_i)) \tag{A.48}$$

Substituting this back in our original calculation, we get:

$$\langle\psi|\psi\rangle \leq 2 - 2\sum_{y\in\Delta}\Re\left(\langle\xi|\,Q_{f(j)}^{\dagger}Q_y\,|\xi\rangle\,\langle\phi_{(y,j)}|\phi_{(y,j)}\rangle\right) \tag{A.49}$$

$$\leq 2 - 2\sum_{y\in\Delta}\Re\left(\langle\xi|\,Q_{f(j)}^{\dagger}Q_y\,|\xi\rangle\,\Pr(y|j)\right) \tag{A.50}$$

$$\leq 2 - 2\sum_{y\in\Delta}\Re\left(\langle\xi|\,Q_{f(j)}^{\dagger}Q_y\,|\xi\rangle\right)\Pr(y|j) \tag{A.51}$$

$$\leq 2\left[1 - \sum_{y\in\Delta}\Re\left(\langle\xi|\,Q_{f(j)}^{\dagger}Q_y\,|\xi\rangle\right)\right]\Pr(y|j) \tag{A.52}$$

$$\leq 2\sum_{y\in\Delta}\left[1 - \Re\left(\langle\xi|\,Q_{f(j)}^{\dagger}Q_y\,|\xi\rangle\right)\Pr(y|j)\right] \tag{A.53}$$

In the second line, we used the definition of $\langle 0^N | R_j^\dagger \Pi_{\mathcal{M}_y} R_j | 0^N \rangle = \langle \phi_{(y,j)} | \phi_{(y,h)} \rangle$ as the probability of getting outcome y in the register $\mathcal{J}$ given that we have projected onto outcome j in register $\mathcal{N}$ according to Section A.4.3.

Next, we use the fact that the real part of the overlap $\langle \xi | Q_{f(j)}^\dagger Q_y | \xi \rangle$ is between -1 and 1, inclusive, because any $Q_y |\xi\rangle$ is a normalized state, and therefore one minus this quantity is at most 2. We also exclude the case where $y = f(j)$, since in that case $Q_{f(j)}^\dagger Q_y = I$ and the overlap is 1, contributing zero to the sum.

$$\begin{aligned} \langle \psi | \psi \rangle &\leq 2 \sum_{y \neq f(j)} 2 \Pr(y|j) && \text{(A.54)} \\ &\leq 4\epsilon && \text{(A.55)} \end{aligned}$$

In the final line above, we use the definition of $\Pr(y|j)$ from Section A.4.3. The actual answer we want is the norm of the difference vector $|\psi\rangle$, which is the square root of the magnitude of the overlap.

$$|||\psi\rangle|| \leq 2\sqrt{\epsilon} \tag{A.56}$$

This completes the proof.

As an additional note, if V is the copy operator, we have $Q_{f(j)}^\dagger Q_y = \delta_{y,f(j)} I_{\mathcal{K}}$. For each $y = f(j)$, the overlap $\langle \xi | Q_{f(j)}^\dagger Q_y | \xi \rangle$ is 1 and contributes zero to the sum of probabilities. For each $y \neq f(j)$, the overlap is exactly 0, and contributes $\Pr(y|j)$ to the sum. Therefore, instead of line A.54 above, we get:

$$\begin{aligned} \langle \psi | \psi \rangle &\leq 2 \sum_{y \neq f(j)} \Pr(y|j) && \text{(A.57)} \\ &\leq 2\epsilon && \text{(A.58)} \end{aligned}$$

And the final error is

$$|||\psi\rangle|| \leq \sqrt{2\epsilon} \tag{A.59}$$

This makes intuitive sense, since the first bound is overly general. It works for any V. If we know V, we can actually make additional assumptions and get a better (smaller) upper bound for the error, in this case by a factor $\sqrt{2}$. □

A.5.2 Error Probability With Projective Measurement

Now what happens if we perform the operator $\hat{Y} = VW$? That is, we perform W to measure some function f and then we extract the useful information using V, but we don't wish to uncompute the results by performing W^{-1}? There are three related reasons we may wish to do this:

1. because W may be practically difficult to perform, and we don't wish to essentially do it twice

2. because we have projectively measured in the register $\mathcal{K}$ so that we can offload some post-processing to a classical computer, and there is no way to uncompute W at that point.

3. the states in $\mathcal{K}$, after having been projected by $\hat{Y}$, are close to classical, and we wish to avoid maintaining them as coherent quantum states

4. the states in $\mathcal{N}$, after having been entangled with $\mathcal{K}$ and projected by $\hat{Y}$, are still "mostly" within their subspaces $\mathcal{L}_j$, and we wish to use states in these subspaces as a resource.

All of these reasons are true for the problem of quantum Fourier state generation, as described in Section 3.5.

These are both practical reasons in that they don't affect the asymptotic resources required by our algorithm. However, they do affect the asymptotic error of our algorithm. In this section, we will calculate this error based on Theorem 9. This is the main original contribution of this work beyond the exposition in [53].

Theorem 10. *Given the setting in Theorem 9 where W approximately measures a function $f : \Omega \to \Delta$ with error ϵ: If the operator $\hat{Y} = VW$ followed by projective measurement on $\mathcal{K}$ yields classical outcome $y \in \Delta$, then the overlap of the state $|\tilde{psi}\rangle$ in $\mathcal{N}$ with $\mathcal{L}_j$ such that $f(j) = y$ is upper-bounded by the number of subspaces $|\Omega|$ times the ϵ.*

$$\sum_{j \in \Omega : f(j) \neq y} \langle \tilde{\psi}| \, \Pi_{\mathcal{L}_j} \, |\tilde{\psi}\rangle \leq |\Omega| \epsilon \tag{A.60}$$

Proof. Consider the probability distribution of states over $\mathcal{N} \otimes \mathcal{J} \otimes \mathcal{K}$. Decomposing the entire system over the subspaces of $\mathcal{N} \bigoplus_{j \in \Omega} \mathcal{L}_j$, within each $\mathcal{L}_j$ there is a probability of ϵ associated with error states $y \in \Delta$ such that $y \neq f(j)$.

$$\sum_{y \neq f(j)} \Pr(y|j) \leq \epsilon \tag{A.61}$$

We can define a total error probability by summing over all values of $j \in Omega$, or $\mathcal{L}_| \in \mathcal{N}$.

$$\sum_{j \in \Omega} \sum_{y \neq f(j)} \Pr(y|j) \leq |\Omega| \epsilon \tag{A.62}$$

However, if we projectively measure y in register $\mathcal{K}$, what is the probability mass associated with "impurities" in register $\mathcal{N}$? That is, we want to find $\sum_{j : f(j) \neq y} \Pr(j|y)$. The maximum contribution to this error for any y is the total error probability above. Therefore, we have:

$$\sum_{j : f(j) \neq y} \Pr(j|y) \leq |\Omega| \epsilon \tag{A.63}$$

This argument is independent of any ancillae. □

BIBLIOGRAPHY

[1] Dorit Aharonov and Amnon Ta-Shma. Adiabatic Quantum State Generation and Statistical Zero Knowledge. page 35, January 2003.

[2] Alfred V. Aho and Krysta M. Svore. Compiling Quantum Circuits using the Palindrome Transform. page 17, November 2003.

[3] Noga Alon and Jehoshua Bruck. Explicit Constructions of Depth-2 Majority Circuits for Comparison and Addition. *SIAM Journal on Discrete Mathematics*, 7(1):1–8, February 1994.

[4] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *arXiv:1206.0758*, June 2012.

[5] Adriano Barenco, Charles Bennett, Richard Cleve, David DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, November 1995.

[6] Robert Beals, Stephen Brierley, Oliver Gray, Aram Harrow, Samuel Kutin, Noah Linden, Dan Shepherd, and Mark Stather. Efficient Distributed Quantum Computing. *Proceedings of the Royal Society*, 469, 2013.

[7] Paul W. Beame, Stephen A. Cook, and H. James Hoover. Log Depth Circuits for Division and Related Problems. *SIAM Journal on Computing*, 15(4):994–1003, November 1986.

[8] Stephane Beauregard. Circuit for Shor's algorithm using 2n+3 qubits. *arXiv:quant-ph/0205095*, May 2002.

[9] David Beckman, Amalavoyal Chari, Srikrishna Devabhaktuni, and John Preskill. Efficient networks for quantum factoring. *Physical Review A*, 54(2):1034–1063, August 1996.

[10] Jan Benhelm, Gerhard Kirchmair, Christian F. Roos, and Rainer Blatt. Towards fault-tolerant quantum computing with trapped ions. *Nature Physics*, 4(6):463–466, April 2008.

[11] Charles Bennett. Logical Reversibility of Computation. *Atomic Energy*, (November), 1973.

[12] Charles Bennett. Time/space trade-offs for reversible computation. 18(4):766–776, 1989.

[13] Dominic W. Berry, Graeme Ahokas, Richard Cleve, and Barry C. Sanders. Efficient Quantum Algorithms for Simulating Sparse Hamiltonians. *Communications in Mathematical Physics*, 270(2):359–371, December 2006.

[14] R. B. Blakestad, C. Ospelkaus, A. P. VanDevender, J. H. Wesenberg, M. J. Biercuk, D. Leibfried, and D. J. Wineland. Improved high-fidelity transport of trapped-ion qubits through a multi-dimensional array. page 16, June 2011.

[15] Alex Bocharov, Yuri Gurevich, and Krysta M. Svore. Efficient Decomposition of Single-Qubit Gates into V Basis Circuits. March 2013.

[16] Alex Bocharov and Krysta M. Svore. A Depth-Optimal Canonical Form for Single-qubit Quantum Circuits. June 2012.

[17] Parsa Bonderson, Sankar Das Sarma, Michael Freedman, and Chetan Nayak. A Blueprint for a Topologically Fault-tolerant Quantum Computer. page 6, March 2010.

[18] Jeffrey Booth. Quantum Compiler Optimizations. June 2012.

[19] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Physical Review A*, 71(2), February 2005.

[20] Michael Bremner, Christopher Dawson, Jennifer Dodd, Alexei Gilchrist, Aram Harrow, Duncan Mortimer, Michael Nielsen, and Tobias Osborne. Practical Scheme for Quantum Computation with Any Two-Qubit Entangling Gate. *Physical Review Letters*, 89(24), November 2002.

[21] Anne Broadbent and Elham Kashefi. Parallelizing Quantum Circuits. *Theoretical Computer Science*, 410, April 2009.

[22] Dan E. Browne, Elham Kashefi, and Simon Perdrix. Computational depth complexity of measurement-based quantum computation. In *Proceedings of the Fifth Conference on Theory of Quantum Computation, Communication, and Cryptography (TQC)*, September 2010.

[23] Jehoshua Bruck. Harmonic Analysis of Polynomial Threshold Functions. *SIAM Journal on Discrete Mathematics*, 3(2):168–177, May 1990.

[24] Byung-Soo Choi and Rodney Van Meter. "$\Theta(\sqrt{n})$"-depth Quantum Adder on a 2D NTC Quantum Computer Architecture. *arXiv:1008.5093*, August 2010.

[25] Juan I Cirac, Peter Zoller, et al. Quantum computations with cold trapped ions. *Physical review letters*, 74(20):4091–4094, 1995.

[26] Richard Cleve and John Watrous. Fast parallel circuits for the quantum fourier transform. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCSoo)*, pages 526–536, 2000.

[27] Raphael Dias da Silva, Einar Pius, and Elham Kashefi. Global Quantum Circuit Optimization. page 50, January 2013.

[28] Christopher M. Dawson and Michael A. Nielsen. The Solovay-Kitaev algorithm. page 15, May 2005.

[29] David DiVincenzo and Panos Aliferis. Effective Fault-Tolerant Quantum Computation with Slow Measurements. *Physical Review Letters*, 98(2), January 2007.

[30] Thomas G. Draper. Addition on a Quantum Computer. *arXiv:quant-ph/0008033*, August 2000.

[31] Thomas G. Draper, Samuel A. Kutin, Eric M. Rains, and Krysta M. Svore. A logarithmic-depth quantum carry-lookahead adder. *Quantum Information and Computation*, 6(3):192–212, June 2006.

[32] Guillaume Duclos-Cianci and Krysta M. Svore. A State Distillation Protocol to Implement Arbitrary Single-qubit Rotations. October 2012.

[33] Bryan Eastin. Distilling one-qubit magic states into Toffoli states. December 2012.

[34] M. Fang, S. Fenner, F. Green, S. Homer, and Y. Zhang. Quantum lower bounds for fanout. *Quantum Information and Computation*, 6:2006, 2003.

[35] R.P. Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982.

[36] Austin G Fowler. Constructing arbitrary Steane code single logical. *Quantum Information and Computation*, 11:867–873, 2011.

[37] Austin G. Fowler, Simon J. Devitt, and Lloyd C. L. Hollenberg. Implementation of Shor's Algorithm on a Linear Nearest Neighbour Qubit Array. *Quantum Information and Computation*, 4:237–251, February 2004.

[38] Ignacio García-Mata, Klaus Frahm, and Dima Shepelyansky. Effects of imperfections for Shors factorization algorithm. *Physical Review A*, 75(5):052311, May 2007.

[39] Brett Giles and Peter Selinger. Exact synthesis of multiqubit Clifford+T circuits. *Physical Review A*, 87(3):032332, March 2013.

[40] Mikael Goldmann and Marek Karpinski. Simulating threshold circuits by majority circuits. In *SIAM Journal on Computing*, pages 551–560, 1994.

[41] Phil Gossett. Quantum Carry-Save Arithmetic. *arXiv:quant-ph/9808061*, 1998.

[42] Frederic Green, Steven Homer, Cristopher Moore, and Christopher Pollett. Counting, Fanout, and the Complexity of Quantum ACC. June 2001.

[43] H Haffner, C Roos, and R Blatt. Quantum computing with trapped ions. *Physics Reports*, 469(4):155–203, December 2008.

[44] Aram Harrow. *Quantum Compiling*. Undergrad thesis, Massachusetts Institute of Technology, May 2001.

[45] Aram Harrow and Austin Fowler. Private communication, Oct 2011.

[46] Aram W. Harrow, Benjamin Recht, and Isaac L. Chuang. Efficient discrete approximations of quantum gates. *J. Math. Phys.*, 43(4445), 2002.

[47] Peter Høyer and Robert Špalek. Quantum Circuits with Unbounded Fan-out. *Theory of Computing*, 1(5):81–103, August 2005.

[48] Cody Jones. Distillation protocols for Fourier states in quantum computing. *arXiv:1303.3066*, March 2013.

[49] Cody Jones. Low-overhead constructions for the fault-tolerant Toffoli gate. *Physical Review A*, 87(2):022328, February 2013.

[50] N. Cody Jones, Rodney Van Meter, Austin G. Fowler, Peter L. McMahon, Jungsang Kim, Thaddeus D. Ladd, and Yoshihisa Yamamoto. A Layered Architecture for Quantum Computing Using Quantum Dots. page 39, October 2010.

[51] N. Cody Jones, James D. Whitfield, Peter L. McMahon, Man-Hong Yung, Rodney Van Meter, Alán Aspuru-Guzik, and Yoshihisa Yamamoto. Simulating chemistry efficiently on fault-tolerant quantum computers. *New Journal of Physics*, (115023), April 2012.

[52] D. Kielpinski, C. Monroe, and D.J. Wineland. Architecture for a large-scale ion-trap quantum computer. *Nature*, 417:709–711, 2002.

[53] A. Yu. Kitaev, A.H. Shen, and M.N. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, Providence, Rhode Island, 2002.

[54] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Asymptotically optimal approximation of single qubit unitaries by Clifford and T circuits using a constant number of ancillary qubits. December 2012.

[55] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single qubit unitaries generated by Clifford and T gates. June 2012.

[56] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Practical approximation of single-qubit unitaries by single-qubit quantum Clifford and T circuits. page 9, December 2012.

[57] Emanuel Knill. An analysis of Bennett's pebble game. page 15, August 1995.

[58] Lucas Kreger-Stickles and Mark Oskin. Microcoded Architectures for Ion-Tap Quantum Computers. *ACM SIGARCH Computer Architecture News*, 36(3):165–176, June 2008.

[59] Samuel A. Kutin. Shor's algorithm on a nearest-neighbor machine. *arXiv:quant-ph/0609001*, August 2006.

[60] Andrew J. Landahl and Chris Cesare. Complex instruction set computing architecture for performing accurate quantum Z rotations with less magic. page 16, February 2013.

[61] Arjen K Lenstra and Hendrik W Lenstra. *The development of the number field sieve*, volume 1554. Springer-Verlag, Berlin, 1993.

[62] Robert Y. Levine and Alan T. Sherman. A Note on Bennetts Time-Space Tradeoff for Reversible Computation. *SIAM Journal on Computing*, 19(4):673–677, August 1990.

[63] Ming Li, John Tromp, and Paul Vitányi. Reversible simulation of irreversible computation. *Physica D: Nonlinear Phenomena*, 120(1-2):168–176, September 1998.

[64] S. Lloyd et al. Universal quantum simulators. *Science*, pages 1073–1077, 1996.

[65] Seth Lloyd. Almost Any Quantum Logic Gate is Universal. *Physical Review Letters*, 75(2):346–349, July 1995.

[66] A. Lubotsky, R. Phillips, and P. Sarnak. *International Communications of Pure and Applied Mathematics*, 40(401), 1987.

[67] Igor Markov and Mehdi Saeedi. Constant-optimized quantum circuits for modular multiplication and exponentiation. *Quantum Information and Computation*, 12(5–6):361–394, 2012.

[68] Ken Matsumoto and Kazuyuki Amano. Representation of Quantum Circuits with Clifford and $\pi/8$ Gates. June 2008.

[69] Rodney Van Meter, W. J. Munro, Kae Nemoto, and Kohei M. Itoh. Arithmetic on a distributed-memory quantum multicomputer. *ACM Journal on Emerging Technologies in Computing Systems*, 3(4):1–23, January 2008.

[70] César Miquel, Juan Paz, and Roberto Perazzo. Factoring in a dissipative quantum computer. *Physical Review A*, 54(4):2605–2613, October 1996.

[71] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L. M. Duan, and J. Kim. Large Scale Modular Quantum Computer Architecture with Atomic Memory and Photonic Interconnects. *arXiv:1208.0391*, August 2012.

[72] P.L. Montgomery. Modular multiplication without trial division. *Math. Computation*, 44:519–521, 1985.

[73] Cristopher Moore. Quantum Circuits: Fanout, Parity, and Counting. March 1999.

[74] Cristopher Moore and Martin Nilsson. Parallel Quantum Computation and Quantum Codes. August 1998.

[75] Michele Mosca and Artur Ekert. The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer. page 16, March 1999.

[76] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, U.K., 2000.

[77] S. Parker and M. Plenio. Efficient Factorization with a Single Pure Qubit and logN Mixed Qubits. *Physical Review Letters*, 85(14):3049–3052, October 2000.

[78] J. O. Rabin and Jeffrey Shallit. Randomized algorithms in number theory. *Communications on Pure and Applied Mathematics*, 39:S239–S256, 1985.

[79] Robert Raussendorf and Hans Briegel. *Physical Review Letters*, 86:5188–5191, 2001.

[80] Robert Raussendorf, Daniel Browne, and Hans Briegel. Measurement-based quantum computation on cluster states. *Physical Review A*, 68(2), August 2003.

[81] John H. Reif and Stephen R. Tate. On Threshold Circuits and Polynomial Computation. *SIAM Journal on Computing*, 21(5):896–908, 1992.

[82] David Rosenbaum. Optimal Quantum Circuits for Nearest-Neighbor Architectures. *8th Conference on Theory of Quantum Computation, Communication and Cryptography*, May 2013.

[83] Jay Sau, Sumanta Tewari, and S. Das Sarma. Universal quantum computation in a semiconductor quantum wire network. *Physical Review A*, 82(5), November 2010.

[84] Peter Selinger. Efficient Clifford+T approximation of single-qubit operators. December 2012.

[85] Vivek V Shende, Stephen S Bullock, and Igor L Markov. Synthesis of Quantum Logic Circuits. *IEEE Transactions on Computer-Aided Design*, 25(6):1000 – 1010, 2006.

[86] V.V. Shende, I.L. Markov, and S.S. Bullock. Smaller two-qubit circuits for quantum communication and computation. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 2, pages 980–985, 2004.

[87] P. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, Santa Fe, NM, November 1994.

[88] K.-Y. Siu, J. Bruck, T. Kailath, and T. Hofmeister. Depth efficient neural networks for division and related problems. *IEEE Transactions on Information Theory*, 39(3):946–956, May 1993.

[89] Kai-Yeung Siu and Jehoshua Bruck. On the Power of Threshold Circuits with Small Weights. *SIAM Journal on Discrete Mathematics*, 4(3):423–435, August 1991.

[90] K.Y. Siu, V.P. Roychowdhury, and T. Kailath. Depth-size tradeoffs for neural computation. *IEEE Transactions on Computers*, 40(12):1402–1412, 1991.

[91] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 77–82, New York, New York, USA, January 1987. ACM Press.

[92] A. Sørensen and K. Mølmer. Quantum computation with ions in thermal motion. *Physical review letters*, 82(9):1971–1974, 1999.

[93] Anders Sørensen and Klaus Mølmer. Entanglement and quantum computation with ions in thermal motion. *Physical Review A*, 62(2):022311, July 2000.

[94] Krysta M. Svore, Matthew B. Hastings, and Michael Freedman. Faster Phase Estimation. April 2013.

[95] Yasuhiro Takahashi and Noboru Kunihiro. A linear-size quantum circuit for addition with no ancillary qubits. *Quantum Information and Computation*, 5(6):440–448, 2005.

[96] Yasuhiro Takahashi and Noboru Kunihiro. A quantum circuit for Shor's factoring algorithm using 2n+2 qubits. *Quantum Information and Computation*, 6(2):184–192, 2006.

[97] Yasuhiro Takahashi and Seiichiro Tani. Constant-Depth Exact Quantum Circuits for the OR and Threshold Functions. December 2011.

[98] Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. Quantum Addition Circuits and Unbounded Fan-Out. *Quantum Information and Computation*, 10(9–10):872–890, October 2010.

[99] Rodney Van Meter. *Architecture of a Quantum Multicomputer Optimized for Shor's Factoring Algorithm*. Ph.d., Keio University, 2006.

[100] Rodney Van Meter and Kohei Itoh. Fast quantum modular exponentiation. *Physical Review A*, 71(5), May 2005.

[101] Rodney Van Meter, Kohei Itoh, and Thaddeus Ladd. Architecture-dependent execution time of Shor's algorithm. *Proceedings of Mesoscopic Superconductivity and Spintronics*, May 2006.

[102] Juha Vartiainen, Mikko Möttönen, and Martti Salomaa. Efficient Decomposition of Quantum Gates. *Physical Review Letters*, 92(17):177902, April 2004.

[103] Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Physical Review A*, 54(1):147–153, July 1996.

[104] C. S. Wallace. A Suggestion for a Fast Multiplier. *IEEE Transactions on Electronic Computers*, EC-13(1):14–17, February 1964.

[105] Mark Whitney, Nemanja Isailovic, Yatish Patel, and John Kubiatowicz. Automated generation of layout and control for quantum circuits. *ACM Computing Frontiers*, 2007.

[106] Mark G. Whitney, Nemanja Isailovic, Yatish Patel, and John Kubiatowicz. A fault tolerant, area efficient architecture for Shor's factoring algorithm. *Proceedngs of the International Symposium on Computer Architecture*, 2009.

[107] Chi-Hsiang Yeh and Emmanouel A. Varvarigos. New Efficient Majority Circuits for the Computation of Some Basic Arithmetic Functions. *Journal of Computing and Information*, 2(1):114–136, 1996.

[108] Christof Zalka. Fast versions of Shor's quantum factoring algorithm. *arXiv:quant-ph/9806084v1*, 1998.

[109] Bei Zeng, Andrew Cross, and Isaac L. Chuang. Transversality Versus Universality for Additive Quantum Codes. *IEEE Transactions on Information Theory*, 57(9):6272–6284, September 2011.