THE UNIVERSITY OF CHICAGO

QUANTUM AND CLASSICAL ALGORITHMS AND OPTIMIZATIONS ENABLING PRACTICAL QUANTUM COMPUTATION

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY

ADAM HOLMES

CHICAGO, ILLINOIS
DECEMBER 2020

This work is dedicated to my family – my mother, father, and sister, who have been a constant source of encouragement and love and have made all of this possible.

I would like to thank my academic advisor Fred Chong. He is an extraordinary scientist, and has been an incredible source of guidance and expertise that has been instrumental in shaping me into the scientist and researcher I am today. He has an uncanny ability to distill immensely sophisticated areas of knowledge down to their core, and address the important questions directly. He has been a wonderful and supportive teacher, mentor, and friend.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This dissertation details six pieces of research spanning architectures and algorithms with an eye toward enabling practical quantum computation. It is separated into two main sections: architectural design and optimizations and near-term quantum algorithm design. The first of these describes work related to the architectural design of quantum computing systems, including optimizations related to single-flux quantum ASICs for error correction decoding [101], optimized compilation for magic-state distillation as a fundamental subroutine of surface code topological quantum error correction [54], and a network architecture for optimized distribution of magic-states within a surface code protected quantum machine [99]. The second section is the design and development of quantum algorithms for near-term, practical-scale quantum computers. It includes design and optimization of representative quantum algorithmic subroutines co-designed with near-term quantum device connectivities [100], as well as a set of novel procedures and quantum algorithms for preparing states corresponding to smooth, differentiable, real-valued functions [102] and analytical proofs of entanglement properties of these families of functions [103]. Applications of these techniques are discussed as well.

Part I

Introduction and Thesis Statement

CHAPTER 1

INTRODUCTION

Quantum computation represents more than just a novel piece of technology – fundamentally, it is a paradigm shift in the way that we think about computation and problem solving altogether. Conceptually, the technology and solutions are promising to offer benefits to many different fields ranging from agriculture, energy, and materials science as well as the newly explored areas of quantum machine learning and combinatorial optimization. This is only part of the promise the technology truly offers though. In a very real sense, the technology itself represents a major leap forward in the progress of human knowledge. If and when we manage to build a large-scale fault-tolerant quantum computer, and design and run meaningful applications on the machine, we will have managed to create a macroscopic classically-controlled piece of microscopic quantum mechanical material. That is a feat on similar ground as the LIGO gravitational wave detection experiments, the construction of the first high-energy particle accelerators, and the breakthroughs of the Manhattan Project. The impacts of such a device can only be speculated on for now, but even with our projections based on our currently limited state of knowledge these devices have the potential to radically change the fields of quantum chemistry through pharmaceutical molecular simulation, thoroughly develop our understanding of natural processes like nitrogen fixation and protein folding, and allow us to design significantly more efficient materials like high-temperature superconductors and solar cells.

Many novel and creative scientific achievements are required to get us to that point. Quantum error correction is a necessary component that entire machines need to be designed around, and these techniques require information theoretic advances, algorithmic theory advances, hardware control advances, and quantum bit materials and fabrication advances. All of these need to be tied together through a cohesive architectural design that leverages each component efficiently, and orchestrates the operation of possibly the most sophisticated, intricate piece of technology humanity has ever constructed.

Equally as important are advancements in our understanding of computation itself. Information processing using quantum mechanical information carriers represents a fundamentally different type of processing than has been developed to date. Many significant achievements have already been made towards building efficient quantum algorithms, and many of these have actually led back to more efficient corresponding classical algorithms. This type of back-and-forth is a very positive effect of quantum research, and illustrates the very concrete impacts of quantum algorithmic research overall.

What has yet to be determined or developed is essentially a foundational understanding of quantum information processing. Many developed algorithms and algorithmic techniques are often counter-intuitive, and one of the major leaps the field has yet to make is to bridge the gap between our intuitive grasp of classical algorithmic problem solving and the many non-intuitive facets of quantum theory. The difficulty in building an intuitive understanding of quantum information processing stems largely from the non-intuitive aspects of quantum theory itself. Understanding quantum theory will clearly lead to understanding quantum information theory at a deeper level, but the opposite direction is one of the most exciting aspects of this field: the possibility that an understanding of quantum information processing can then inform a deeper understanding of quantum theory.

This dissertation covers six pieces of research along both of these major directions: error-corrected quantum system architectural design and quantum algorithms and information theory. It represents our attempts to push the field further, and to do our part to enable this landmark human achievement. Hopefully this work inspires more creative ideas, and serves as a shoulder for others to stand on as we collectively continue on our journey to achieve this goal.

CHAPTER 2

THESIS STATEMENT

Quantum and classical algorithmic and architectural design is fundamental to enabling the construction of useful near-term and fault tolerant quantum computers. This thesis demonstrates that careful design of error correcting architectures and protocols can significantly reduce the barriers and overheads of constructing error-corrected fault tolerant machines. Additionally, careful design of algorithmic and information theoretic methods to encode classical data into quantum computers results in more efficient algorithm execution, a necessity in both the near-term and fault tolerant regimes. Altogether, this work designs and evaluates techniques that help to enable useful quantum computation in the near-term, and seek to bring fault-tolerant quantum computation closer to reality.

Part II

Architectural Designs and Optimizations

CHAPTER 3

NISQ+: BOOSTING COMPUTATIONAL POWER OF QUANTUM COMPUTERS BY APPROXIMATING QUANTUM ERROR CORRECTION

Quantum computers are growing in size, and design decisions are being made now that attempt to squeeze more computation out of these machines. In this spirit, we design a method to boost the computational power of near-term quantum computers by adapting protocols used in quantum error correction to implement "Approximate Quantum Error Correction (AQEC)." By approximating fully-fledged error correction mechanisms, we can increase the compute volume (qubits $\times$ gates, or "Simple Quantum Volume (SQV)") of near-term machines. The crux of our design is a fast hardware decoder that can approximately decode detected error syndromes rapidly. Specifically, we demonstrate a proof-of-concept that approximate error decoding can be accomplished online in near-term quantum systems by designing and implementing a novel algorithm in superconducting Single Flux Quantum (SFQ) logic technology. This avoids a critical decoding backlog, hidden in all offline decoding schemes, that leads to idle time exponential in the number of T gates in a program [206].

Our design utilizes one SFQ processing module per physical quantum bit. Employing state-of-the-art SFQ synthesis tools, we show that the circuit area, power, and latency are within the constraints of typical, contemporary quantum system designs. Under a pure dephasing error model, the proposed accelerator and AQEC solution is able to expand SQV by factors between 3,402 and 11,163 on expected near-term machines. The decoder achieves a 5% accuracy threshold as well as pseudo-thresholds of approximately $5\%, 4.75\%, 4.5\%$, and 3.5% physical error rates for code distances $3, 5, 7$, and 9, respectively. Decoding solutions are achieved in a maximum of ~ 20 nanoseconds on the largest code distances studied. By avoiding the exponential idle time in offline decoders, we achieve a 10x reduction in required code distances to achieve the same logical performance as alternative designs.

3.1 Introduction

Quantum computing has the potential to revolutionize computing and have massive effects on major industries including agriculture, energy, and materials science by solving computational problems that are intractable with conventional machines [96, 201]. As we begin to build quantum computing machines of between 50-100 qubits [178] and larger, design decisions are being made to attempt to get the most computation out of a machine, quantified in this work by expanding the "Simple Quantum Volume" (SQV). SQV can be defined as the number of computational qubits of a machine multiplied by the number of gates we expect to be able to perform without error, as in Figure 3.1. One limiting factor on SQV now is that physical quantum bits (qubits) are extremely error-prone, which means that computation on these machines is bottlenecked by the short lifetimes of qubits. System designers combat this by attempting to build better physical qubits, but this effort is extremely difficult and classical systems can be used to alleviate the burden. Specifically, *quantum error correction* is a classical control technique that decreases the rate of errors in qubits and expands the SQV. Error correction proceeds by encoding a set of *logical* qubits to be used for algorithms into a set of faulty physical qubits. Information about the current state of the device, called *syndromes*, is extracted by a specific quantum circuit that does not disturb the underlying computation. Decoding is the process by which an error correcting protocol maps this information to a set of *corrections* that, if chosen correctly, should return the system to the correct logical state. Fully fault tolerant machines can expand the SQV rapidly by suppressing qubit errors exponentially with the code distance.

While a fully fault-tolerant quantum computer may take many years to construct, it is possible to use the well-developed theory of error correction as inspiration for constructing *error mitigation* protocols that still provide a strong expansion in SQV. In this paper we present an approximate decoding solution specifically targeting execution time and show that we can in fact perform decoding at the speed of syndrome generation for near-term machines. Prior work has suggested and analyzed software solutions for decoding, but

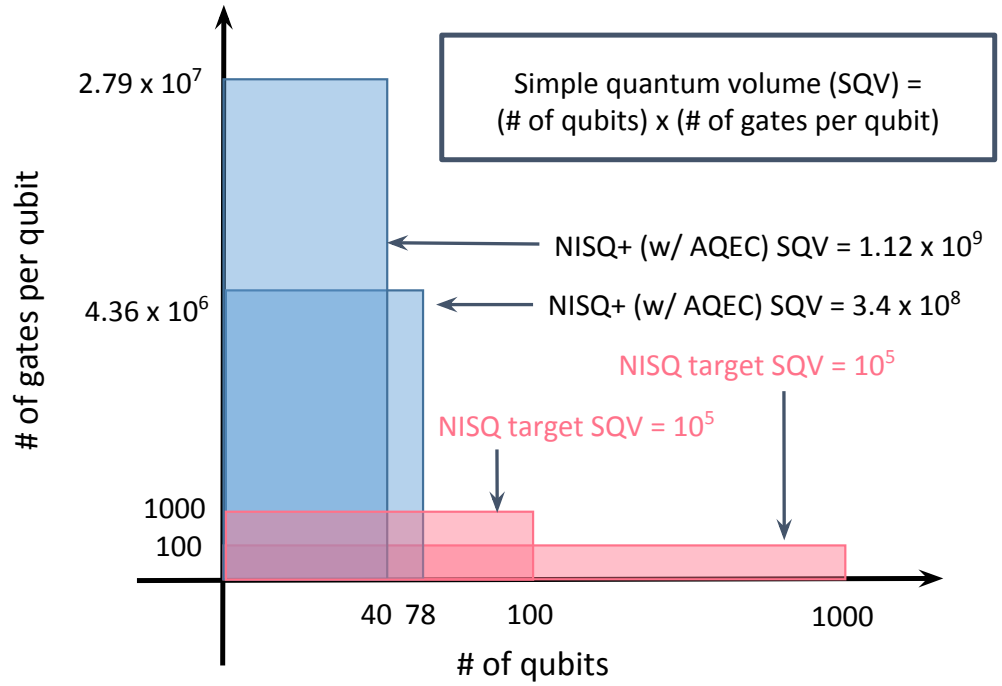


Figure 3.1: Boosting the quantum computation power with approximate error correction schemes. A machine with 1024 faulty physical qubits of error rate 10^{-5} has an SQV of $\approx 10^8$. By performing fast, online, approximate decoding, we can trade the number of computational qubits for gate fidelity and boost the SQV by over a factor of 3,402. Moving to a higher code distance raises this increase to a factor of 11,163. NISQ machines are severely limited by gate fidelity, and introducing error mitigation techniques can have dramatic effects on SQV.

relying on hardware-software communication can be slow, especially considering the cryogenic environment of typical quantum computing systems. If decoding occurs slower than error information is generated, the system will generate a backlog of information as it waits for decoding to complete, introducing an exponential time overhead that will kill any quantum advantage (see Section 3.3). A hardware solution proposed here results in the ability to perform logical gates with orders of magnitude better fidelity and at the speed of syndrome generation, resulting in a major expansion in SQV as shown in Figure 3.1. This relies on an approximate decoding algorithm implemented in superconducting Single Flux Quantum (SFQ) hardware. While the algorithmic design enables the accuracy of the hardware accelerator to be competitive at small scale with existing software implementations, the benefits of implementing the circuitry directly in SFQ hardware are numerous. Specifically, high clock speeds, low power dissipation, and unique gating style allows for our accelerator to be co-located with a

quantum chip inside a dilution refrigerator, avoiding otherwise high communication costs.

This work contributes the following:

1. We design an approximate decoding algorithm for stabilizer codes based on SFQ hardware, leveraging unique capabilities that the hardware offers,

2. We show that using this new error mitigation technique, we can expand the SQV of near-term machines by factors of between 3,402 and 11,163,

3. We use Monte-Carlo simulation based benchmarking of the hardware accelerator, resulting in effective accuracy and pseudo-thresholds,

4. We perform system execution time analysis, realistically benchmarking the decoder performance in real time and showing that decoding is likely to be able to proceed at or exceeding the speed of data generation enabling the benefits of fault tolerant quantum computing.

5. We show that our online decoder requires 10x smaller code distance than offline decoders when decoding backlog accounted for.

The remainder of the paper is as follows: Section 3.2 describes the necessary background of quantum computation and details the specifications of typical quantum computing systems stacks. Section 3.2.2 describes quantum error correction and the decoding problem in detail. Section 7.2 describes relevant related work in the area ranging from optimized software implementations of matching algorithms to novel descriptions of neural network based decoders. Section 3.5 describes our decoding algorithm, and Section 3.6 describes implementation details of SFQ technology, and the circuit datapaths in detail. Section 3.7 describes our methodology for evaluation, including details of the simulation environment in which our accelerator was benchmarked, details of the metrics used to evaluate performance, and descriptions of novel synthesis tools used to generate efficient layouts of SFQ circuitry. Section 3.8 presents our accuracy results, a breakdown of the accelerator characterization

including area, power, and latency footprints, a timing evaluation, and analysis of the SQV effects. Section IV concludes.

3.2 Background

In this section we discuss the basics of quantum computation, quantum error correction, and a description of the fundamental components of a quantum computing system architecture.

3.2.1 Basics of Quantum Computation

Here we provide a brief overview of quantum computation necessary to discuss quantum error correction. For more detailed discussions see [154]. A quantum computing algorithm is a series of operations on two level quantum states called *qubits*, which are quantum analogues to classical bits. A qubit state can be written mathematically as a superposition of two states as $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, where the coefficients $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. A measured qubit will yield a value of $|0\rangle$ or $|1\rangle$ with probability $|\alpha|^2$ or $|\beta|^2$, respectively, at which point the qubit state will be exactly $|0\rangle$ or $|1\rangle$. Larger quantum systems are represented simply as $|\psi\rangle = \sum_i \alpha_i\,|i\rangle$ where $|i\rangle$ are computational basis states of the larger quantum system.

Quantum operations (gates) transform qubit states to other qubit states. In this work we will be making use of particular quantum operations known as *Pauli gates*, denoted as $\{I, X, Y, Z\}$. These operations form a basis for all quantum operations that can occur on a single qubit, and therefore any operation can be written as a linear combination of these gates. Additionally, error correction circuits make use of the Hadamard gate H, an operation that constructs an evenly weighted superposition of basis elements when acting on a basis element. Two-qubit controlled operations will also be used, which can generate entanglement between qubits and are required to perform universal computation.

3.2.2 Quantum Error Correction

Qubits are intrinsically fragile quantum systems that require isolation from environmental interactions in order to preserve their values. *Decoherence*, for example the decay of a quantum state from a general state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ to the ground state $|\psi'\rangle = |0\rangle$ happens rapidly in many physical qubit types, often on the order of tens of nanoseconds [207, 205]. This places a major constraint on algorithms: without any modifications to the system, algorithms can only run for a small, finite time frame with high probability of success.

To combat this, *quantum error correction* protocols have been developed. These consist of encoding a small number of *logical* qubits used for computation in algorithms into a larger number of physical qubits, resulting in a higher degree of reliability [140, 53, 74, 206]. In general, developing quantum error correction protocols is difficult as directly measuring the qubits that comprise a system will result in destruction of the data. To avoid this, protocols rely upon indirectly gathering error information via the introduction of extra qubits that interact with the primary set of qubits and are measured. This measurement data is then used to infer the locations of erroneous data qubits.

While many different types of protocols have been developed, this work focuses primarily on the *surface code*, a topological stabilizer code [88] that is widely considered to be the best performing code for the medium-term as it relies purely on geometrically local interactions between physical qubits greatly facilitating its fabrication in hardware, and has been shown to have very high reliability overall [74].

3.2.3 The Surface Code

Errors can occur on physical qubits in a continuous fashion, as each physical qubit is represented mathematically by two complex coefficients that can change values in a continuous range. However, a characteristic of the quantum mechanics leveraged by the surface code is that these continuous errors can be *discretized* into a small set of distinct errors. In particular, the action of the surface code maps these continuous errors into Pauli error operators of the

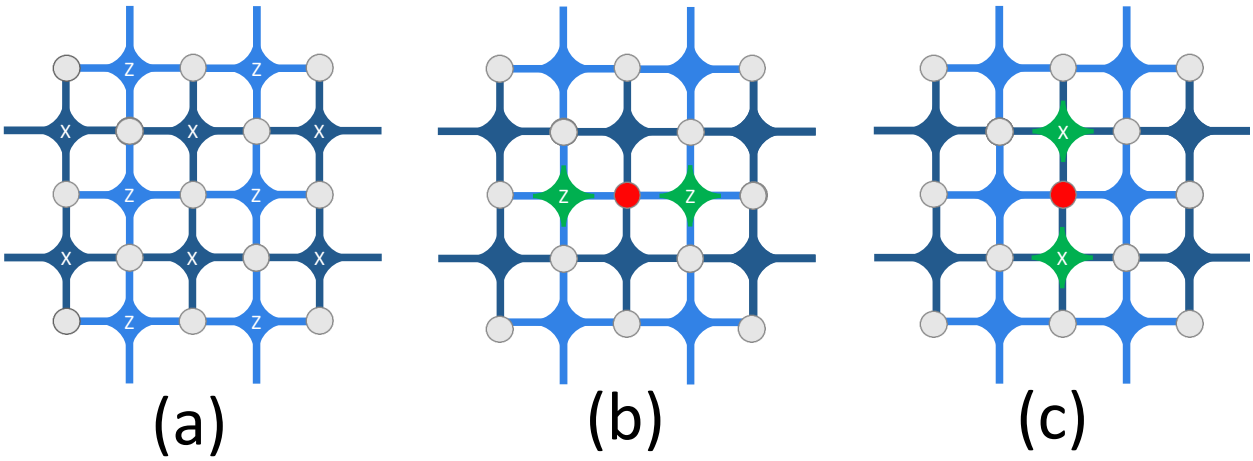


Figure 3.2: Figure (a) shows a graphical illustration of a surface code mesh. Gray circles indicate data qubits, and nodes labeled X and Z indicate ancillary qubits measuring X and Z stabilizers, respectively. Ancillary qubits are joined by colored edges to the data qubits that they are responsible for measuring. In figure (b) a single data qubit experiences a Pauli X error indicated by red coloring, causing the neighboring Z ancillary qubits to detect an odd parity in their data qubit sets and return $+1$ measurement values indicated by green coloring. In figure (c), the data qubit in red experiences a Pauli Z error, causing the vertically adjacent X ancillary qubits to return $+1$ measurement values. The entire error syndrome strings for either of these two cases would include a string of 12 values, two of which would be $+1$ and the remaining 10 would be 0.

form $\{I, X, Y, Z\}$ occurring on the data. This is one of the main features of the code that allows error detection and correction to proceed.

The surface code procedure that accomplishes error discretization, detection, and correction is an error correcting code that operates upon a two-dimensional lattice of physical qubits. The code designates a subset of the qubits as data qubits responsible for forming the logical qubit, and others as ancillary qubits responsible for detecting the presence of errors in the data. This is shown graphically in Figure 3.2. Ancillary qubits interact with all of their neighboring data qubits and are then measured, and the measurement outcomes form the *error syndrome*. This set of operations forms the *stabilizer circuit*, where each ancillary qubit measures a four-qubit operator called a *stabilizer*.

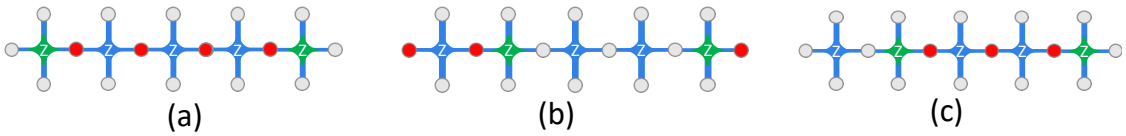


Figure 3.3: Figure (a) shows a data qubit error pattern spanning across ancillary qubits. Each data qubit experiencing error is indicated in red, and the ancillary qubits returning +1 measurement values are indicated in green. Each ancillary qubit that is adjacent to two erroneous data qubits does not signal the presence of any errors, as the parity of the data qubit sets are still even. This creates an *error string* that runs from the ancillary qubit on the left of the grid to the one on the right. Decoding must map these +1 values to the corresponding set of 4 data qubit errors that generated it. Figures (b) and (c) show degeneracy in error syndrome generation by surface code data qubit error patterns. The figures depict two distinct sets of data qubit error patterns that both generate the same error syndromes. Both patterns contain the same number of physical data errors, so these patterns are equally likely assuming independence of errors.

Error Detection

The ancillary qubits are partitioned into those denoted as X and Z ancilla qubits. These ancilla qubit sets are sufficient for capturing any Pauli error on the data qubits, as Y operators can be treated as a simultaneous X and Z error. The action of the X stabilizer is two-fold: the four neighboring data qubits are forced into a particular state that discretizes any errors that may have occurred on them. Second, the measurement of the X ancilla qubit signals the parity of the number of errors that have occurred on its four neighbors. For example, it yields a $+1$ value if the state of the four neighboring qubits has an even number of Z errors. The same is true of the Z stabilizers – these track the parity of X errors occurring in the neighboring qubits. If an odd number of errors have occurred in either case, the ancilla qubit measurement will yield a $+1$ value, an event known as a *detection event* [77], otherwise these will return values of 0 or -1 depending on convention. We will refer to the ancillary qubits returning $+1$ values as *hot syndromes*. The *error syndrome* of the code is a bit string of length equal to the total number of ancilla qubits, and is composed of all of these measurement values.

Decoding is the process of mapping a particular error syndrome string to a set of corrections to be applied on the device. An example of this process is shown graphically in Figure 3.2.

In this example, the hot syndromes generated by a single data qubit error are marked in red. Each single data qubit error causes the adjacent ancillary qubits to return +1 values.

A different situation occurs when strings of data qubit errors cross ancillary qubits, as shown in Figure 3.3. Here, four consecutive data qubits experience errors which generates hot syndrome measurements on the far left and right of the grid. This is because each ancillary qubit along this chain detects even error parity, so they do not signal the presence of errors. Decoding must be able to pair the two hot syndromes, applying corrections along the chain that connects them.

Error Detection Can Fail

Notice that in Figure 3.3 (a), if the data qubits on the left and right endpoints of the chain had also experienced errors, none of the ancillary qubits would have detected the chain. This represents a class of undetectable error chains in the code, and specifically occurs when chains cross from one side of the lattice to the other. The result of these chains are physical errors present in the code that cannot be corrected, and are known as *logical errors*, as they have changed the state of the logical qubit. One important characteristic of the surface code is the minimal number of qubits required to form a logical error. This number is referred to as the *code distance, d* of a particular lattice.

3.2.4 Quantum Computing Systems Organization

While qubits are the foundation of a device, a quantum computer must contain many layers of controlling devices in order to interact with qubits. Qubits themselves can be constructed using many different technologies, some of which include superconducting circuits [143, 80, 81, 18, 124], trapped ions [149, 143, 68, 94, 135], and quantum dots [231]. Controlling these devices is often performed by application of electrical signals at microwave frequencies [46, 230, 167, 176].

This work focuses on systems built around qubits that require cryogenic cooling to

	# qubits	# total gates	# T gates
takahashi_adder	40	740	266
barenco_half_dirty_toffoli	39	1224	504
cnu_half_borrowed	37	1156	476
cnx_log_depth	39	629	259
cuccaro_adder	42	821	280

Table 3.1: Characteristics of the simulated benchmarks.

milliKelvin temperatures [105]. These systems require the use of dilution refrigerators, and typical architectures involve classical controllers located in various temperature stages of the system. Such a system is described schematically in [205, 105], and presents many design constraints. Controllers inside the refrigerator are subject to area and power dissipation constraints [173, 193]. Communication between stages can be costly as well. Many systems are constructed today using control wiring that scales linearly with the number of qubits, which will prohibit the construction of scalable machines [78].

3.2.5 Classical Control in Quantum Computing Systems

Error correction classical processing requires high bandwidth communication of the measurement values of many qubits on the quantum substrate repeatedly throughout the operation of the device, encouraging studies of engineering solutions [222], feasibility [204] and controller design [205]. Not only are instruction streams primarily dominated by quantum error correction operations [138, 137], but also the classical controller responsible for error correction processing must be tightly coupled to the quantum substrate. If communicating between the quantum substrate and error correcting controller is subject to excessive latencies, the execution of fault tolerant algorithms will be completely prohibited.

3.3 Motivation: Decoding Must be Fast

Decoding must be done quickly for the surface code to perform well. During actual computation on a surface code error corrected device, there exist gates called T-gates that require

knowledge of the current state of errors on the device before they can execute. [1] If decoding is slower than the rate at which syndromes are generated, an algorithm will create a *data backlog*. While the machine is waiting for decoder to process the backlog, more syndrome data is accumulating on the device, which must be processed before executing the subsequent T-gate. Over time, this results in latency overhead that is exponentially dependent upon the number of such gates. Specifically, the overhead scales as $(\frac{r_{\text{gen}}}{r_{\text{proc}}})^k = f^k$, where r_{gen} is the rate of data generation, r_{proc} is the rate of decoder processing, each in bauds, f is the decoding ratio, and k is the number of T gates in the quantum algorithm. An exponentially slow quantum computer eliminates all of its usefulness.

Figure 3.4 shows the exponential latency overhead due to data backlog. The proof of this is summarized as follows (for more details see [206]): suppose $f > 1$. This implies that there will be a time t_0 in the application where we encounter a T gate and must wait for syndrome data to be decoded before continuing. Let Δ_{gen} be the amount of time that the machine must stall for processing this data. During this time an additional $D_1 = r_{\text{gen}} \times \Delta_{\text{gen}}$ bits of syndrome data is generated, which can be processed in time $\Delta_{\text{proc}} = r_{\text{gen}}\Delta_{\text{gen}}/r_{\text{proc}} = f\Delta_{\text{gen}}$. The backlog problem begins to be noticeable at this point, where during processing of the first block D_1, we generate a *new block* $D_2 = r_{\text{gen}} \times \Delta_{\text{proc}} = fD_1 > D1$ in size. Then, at the next T gate this process repeats, and we again generate a block of data of size $D_3 = fD_2 = f^2D_1$ bits. Hence, by the k'th T gate, we generate an overhead of f^kD_1 bits to process, exponential in *the decoder's performance ratio*.

As a specific example, consider a multiply-controlled NOT operation on 100 logical qubits from [100]. This algorithm contains $\sim$ 2356 gates, of which 686 are T-gates after decomposition. Assuming that a syndrome generation cycle time is approximately 400 ns [84], and the best prior decoder requires 800 ns to execute [42], the ratio $(r_{\text{gen}}/r_{\text{proc}}) = 2$, and the execution time is intractable.

Figure 3.5 shows a simulation of real quantum subroutines each composed of a different

1. Errors commute and can be post-corrected for other gates, but not T-gates.

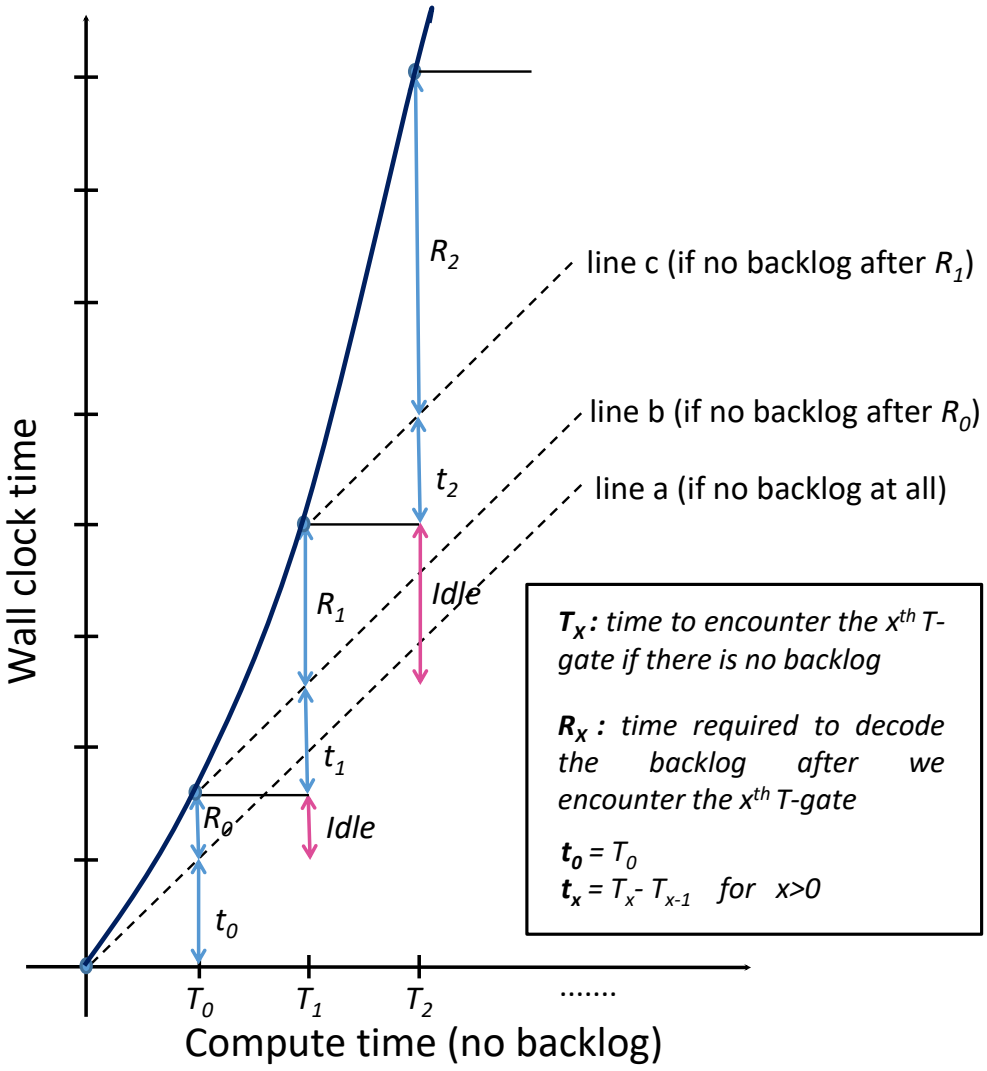


Figure 3.4: Exponential latency overhead when $f = (\frac{r_{\text{gen}}}{r_{\text{proc}}}) > 1$. X-axis shows the compute time if there is no backlog and y-axis shows the actual wall clock time; if there is no backlog we expect wall clock time to be the same as the compute time (line a). Every time we encounter a T-gate we need to decode all the syndromes up until that gate before we can continue the execution [206]. When we encounter the first T-gate at time T_0, we need to finish the decoding of the data generated during t_0 (not all the data is already decoded as decoding rate is slower than data generation rate) and it takes R_0 to do that. During R_0 where our quantum system is idle, more syndromes are generated and when we encounter the second T-gate at $T_1 + R_0$, we need to finish decoding those syndromes in addition to the syndromes generated during t_1 before continuing the program execution. **The syndrome data generated during the idle periods is the key reason behind data backlog creation which leads to exponential latency overhead.**

number of T gates as denoted in Table 3.1. The exponential overhead scaling shows that as decoders become slower than the rate at which data is being generated (which occurs

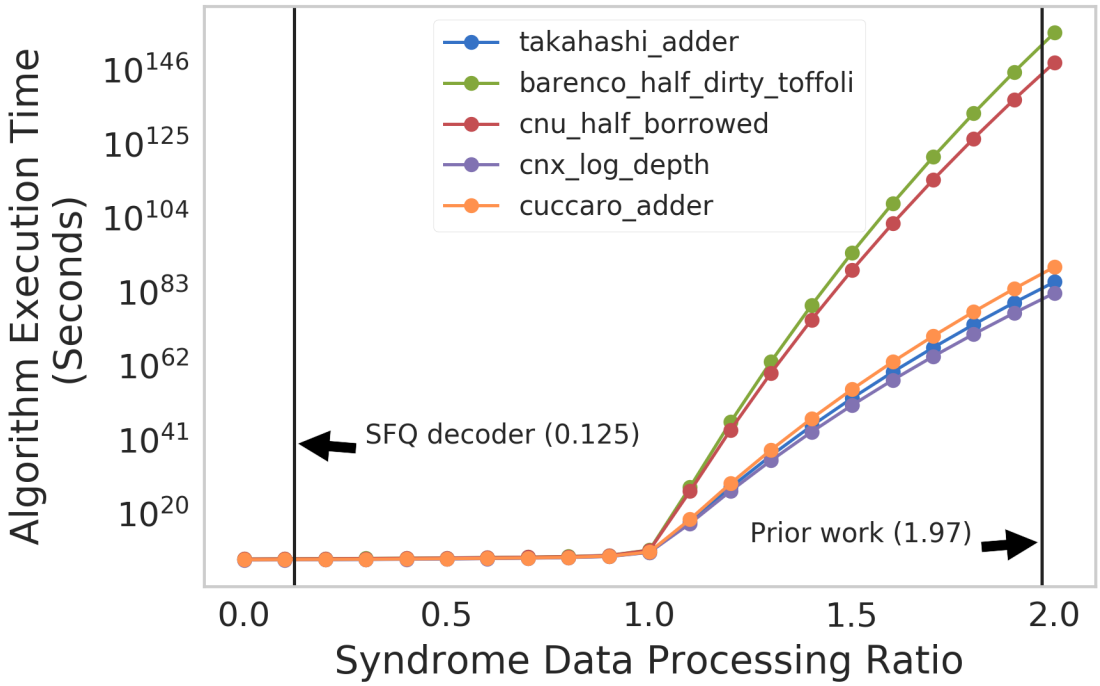


Figure 3.5: Running times of fault tolerant quantum algorithms with decoders of varying efficiency. The X-axis plots $\frac{r_{\text{gen}}}{r_{\text{proc}}}$. To the left of 1, data is processed as fast as it is generated, whereas rates to the right of 1 indicate that the decoder is slower than syndrome data is generated. The T-gates require synchronization with the decoder in order to execute. Prior work [42] claims that fast neural network inference decoders can perform inference in $\sim$ 800 ns, which places the decoder at approximately the 1.5 - 2 region for a system generating syndromes in the 400-500ns range. Our decoding results show that time to solution never exceeds 20ns, placing it below 1. Clearly computation becomes intractable quickly for slow decoders.

for "syndrome data processing ratios" over 1), the overheads quickly become intractable. Regardless of the effectiveness of the decoder, if it operates at a processing ratio higher than 1 then it will impose exponentially high latency overheads on algorithm execution. The algorithms all draw inspiration from [15]. Barenco-half-dirty-Toffoli is a logarithmic depth multi-control Toffoli gate using $\mathcal{O}(n)$ ancilla bits. It performs the same computation as the "cnx-log-depth" gate with a different circuit. The "cnu-half-borrowed" gives an implementation of a multi-control Toffoli using $\mathcal{O}(n)$ dirty ancilla, meaning the initial states of these bits does not need to be known. The Cuccaro adder is a linear depth implementation of a reversible A + B adder, i.e. two registers of the specified length added together. It has a carry in and

a carry out bit as well. The Takahashi adder is an optimized version of the Cuccaro adder [202].

This is the primary motivation for this work – the hardware decoder must be able to execute faster than syndrome data are generated as a prerequisite for tractable fault tolerant computation.

3.4 Related Work

Early work focused on the development of and modifications to the minimum-weight perfect matching algorithm (MWPM) [65, 64] to adapt it to surface code decoding [75, 76]. This resulted in a claimed constant time algorithm after parallelization [70].

Other work has constructed maximum likelihood decoders (MLD) based on tensor network contraction [37]. This work is computationally more expensive than minimum-weight perfect matching, but is more accurate.

Neural networks have been explored as possible solutions to the decoding problem as well [214, 216, 215, 212, 42, 217, 213, 14, 208]. Feed-forward neural networks and recurrent neural networks have been explored in combination with lookup tables to form decoders. The primary distinguishing factor in these systems is that the networks function as *high level decoders* in that they predict both a sequence of error corrections on data qubits along with the existence of a logical error. In this sense, they operate at a higher level than both the MWPM and MLD decoders, seemingly at the cost of execution time with respect to training complexity.

Lastly, more customized algorithms have been developed specifically targeting the surface code decoding problem, including renormalization group decoders [62], union-find decoding [51, 50], and others [229, 61].

The primary distinguishing factor of our work is that the decoder design is guided by practical system performance. Accuracy has been sacrificed in order to achieve quantum advantage. While the proposed decoder design may not achieve logical

error suppression at the same order as some other algorithms, the ability to perform the algorithm in SFQ hardware at or exceeding the speed of syndrome generation is achieved, as is satisfaction of system design constraints.

3.5 Decoder Overview and Design

In this section we describe decoding in terms of a maximum-weight matching problem, followed by details of our approximate decoding algorithm, and demonstrate how we make efficient use of unique features of SFQ gates to implement the algorithm in hardware.

3.5.1 Maximum Weight Matching Decoding

The decoding problem requires that the maximally likely set of error chains be reported as a solution, given a particular error syndrome. This can be formulated as a matching problem. Specifically, given an error syndrome string $S \in \{-1, 1\}^n$, we can construct a complete graph on vertices associated with each ancillary qubit that reported an error. The weight of each edge between vertices is proportional to likelihood of a path between these ancillary qubits on the original surface code grid graph. The goal is therefore to find the maximally likely pairing of the syndromes using these weights, one method for doing so is to solve a maximum-weight perfect matching problem.

3.5.2 A Greedy Approach

Our decoding algorithm is based upon a greedy approximation to the maximum-weight matching problem. The algorithm calculates all distances $d(v_i, v_j)$ between vertices and sorts them in ascending order $d_1, d_2, ..., d_{k'}$ where $k' = \binom{k}{2}$. All of the corresponding probability weights are calculated, transforming this ordering to a descending order of likelihood. Then, for each edge e in descending order, add e to the solution M if it forms a matching. This means that it adds another two distinct vertices into M that were not already present. To

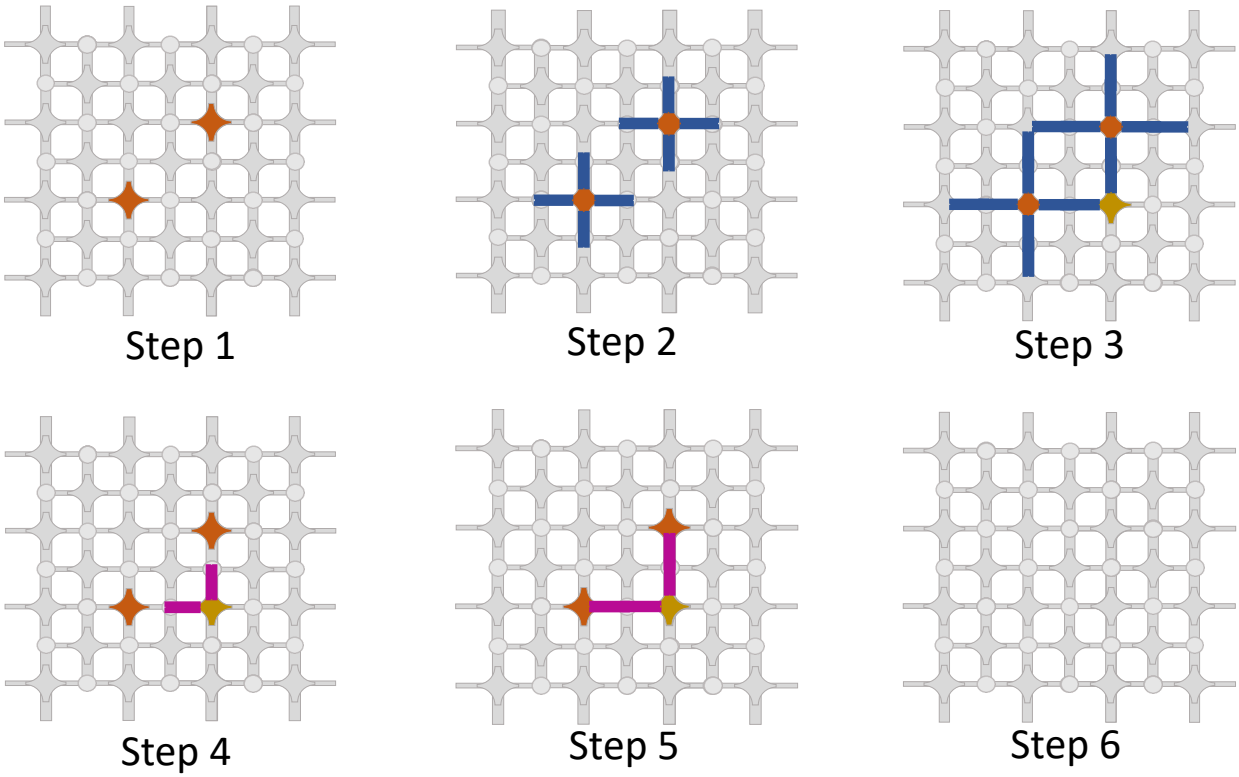


Figure 3.6: Baseline solution to find the two closest hot syndrome modules. Step1: two decoder modules have "1" hot syndrome input. Step2: the hot syndrome modules propagate grow signals. Step3: the grow signals meet at an intermediate module. Step4: the intermediate module sends pair signals in the opposite direction. Step5: pair signals arrive at the hot syndrome modules. Step6: decoding is complete. Note that the decoder modules that receive a pair signal are considered as part of the error chain that has occurred.

account for boundary conditions, we introduce a set of external nodes connected to the appropriate sides of the lattice, and connected to one another with weight 0. Under this formulation, the algorithm is a 2-approximation of the optimal solution [59].

3.5.3 SFQ-Based Decoder

In this section, we introduce the functional design of our SFQ-based decoder and give some rational for each aspect of its design. As a reminder, Single Flux Quantum is classical logic implemented in superconducting hardware that does not perform any quantum computation. It is a medium used to express our classical algorithm. The decoder is placed above the

quantum chip layer; it receives measurement results from ancillary qubits as input, and returns a set of corrections as output. For scalability, our decoder design is built out of a two dimensional array of modules implemented in SFQ logic circuits that we refer to as *decoder modules*. These are connected in a rectilinear mesh topology. Modules are identical and there is one module per each data and ancillary qubit, denoted as *data qubit modules* and *ancilla qubit modules*, respectively. Each decoder module has one input called the *hot syndrome input* that comes from the measurement outcome of the physical quantum bits and determines if the module corresponds to a hot syndrome (note that this input can be "1" only for ancilla qubit modules). Each module contains one output called the *error output* that determines if the module is contained in the error chain (this output can be "1" for all of the decoder modules). In addition, each module has connections to adjacent modules (left, right, up and down).

Our approximate decoder algorithm proceeds as follows. First, the algorithm finds the two modules with "1" hot syndrome input, called *hot syndrome modules*, that are closest together. Next, the algorithm reports the chain of modules connecting them as the correction chain. Finally, it resets the hot syndrome input of the two modules and searches for the next two closest hot syndrome modules. The decoder continues this process until no module with "1" hot syndrome input exists. This is graphically displayed in Figure 3.6.

Baseline Solution: Our baseline design finds the two closest hot syndrome modules as shown in Figure 3.6 as follows: 1) every hot syndrome module sends *grow* signals to all the adjacent modules in all four directions; each adjacent module propagates the grow signal in the same direction. Grow signals propagate one step at each cycle. 2) When two grow signals intersect at an *intermediate module*, we generate a set of *pair* signals and back-propagate these to their hot syndrome origins. All of the decoder modules that receive pair signals are part of the error chain. Note that more than one intermediate module might exist, however, only one of them is effective and sends the pair signals. For example, in Figure 3.6, two intermediate modules receive the grow signals, and the decoder is hardwired to be

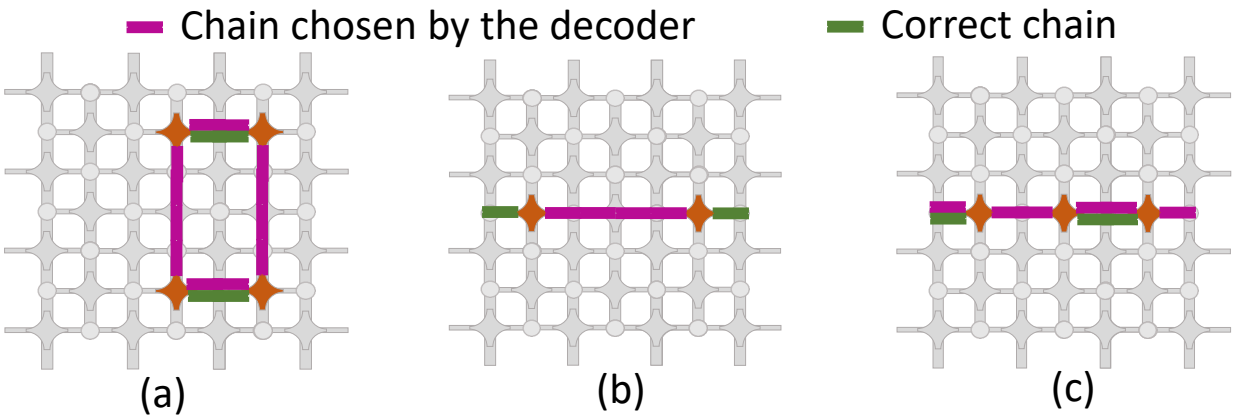


Figure 3.7: Scenarios where the SFQ decoder chooses the wrong chain where (a) no reset/boundary/equidistant mechanisms are employed, (b) no boundary/equidistant mechanisms are employed, and (c) no equidistant mechanism is employed.

effective (ineffective) when it receives grow signals from up and left directions (down and right directions). Intermediate module refers to the effective one. The baseline solution does not show accuracy or pseudo-threshold behavior and demonstrates poor logical error rate suppression, see the incremental results presented in Section 3.8 in Figure 4.10.

Reset Mechanism: One flaw of the baseline system is the lack of a mechanism to reset the decoder modules after two hot syndrome modules are paired. Grow signals of the paired modules continue to propagate, potentially causing these modules to pair incorrectly with other hot syndrome modules, ultimately resulting in an incorrect error chain reported. Figure 3.7 (a) shows an incorrect matching due to this behavior. To mitigate this, we add a reset mechanism that resets the decoder modules each time hot syndrome modules are paired and the error chain connecting them is determined. Adding the reset mechanism to the baseline system improves the performance somewhat, but does not yet achieve tolerable accuracy.

Boundary Mechanism: Another explanation for the low performance of the baseline solution is that it never pairs hot syndrome modules with boundaries. For example, if two hot syndrome modules are far from each other but are close to boundaries, the error chain with the maximum likelihood is the one that connects the hot syndrome modules to the boundaries. Figure 3.7 (b) shows this behavior occurring on a machine. We implement a

mechanism that enables pairing the hot syndrome modules with boundaries. To do this, we add decoder modules that surround the surface boundaries called *boundary module* (one per each quantum bit located at a boundary). Our solution treats boundary modules as hot syndrome modules but they do not grow and can pair only with non-boundary modules. Note that when two modules are paired, the hot syndrome input of only the non-boundary modules is reset; boundary modules are always treated as hot syndrome modules. Adding the boundary mechanism to the baseline solution augmented with the reset mechanism further increases the accuracy of the decoder.

Equidistant Mechanism: Finally, the last major reason for inefficiency of the baseline is that it does not properly handle the scenarios in which multiple hot syndrome modules are spaced within equal distances of one another, resulting in a set of pairs that are all equally likely. The baseline solution augmented with reset and boundary mechanisms works properly only if no non-boundary hot syndrome module has an equal distance to more than one other hot syndrome module; otherwise the solution pairs it with all the hot syndrome modules with equal distance. However, this is not the desired output. We need a more intelligent solution to break the tie in the aforementioned scenario, and pair the hot syndrome module to only one other module. This is shown in Figure 3.7 (c).

To resolve these equidistant degenerate solution sets, we introduce a request – grant policy that allows for the hardware to choose specific subsets of these pairs to proceed. 1) Similar to the baseline solution, the non-boundary hot syndromes first propagate grow signals. 2) An intermediate module receives two grow signals from two different directions, and it sends *pair_request* signals in the opposite directions. Pair_request signals continue to propagate until they arrive at a module with "1" hot syndrome input. 3) The modules with "1" hot syndrome input send *pair_grant* signals in the opposite direction of the received pair_request signals. Note that multiple pair_request signals might arrive at a module with "1" hot syndrome at the same time, but it gives grant to only one of them. 4) An intermediate module receives pair_grant signals from two different directions and sends pair signals in

the opposite directions. 5) Pair signals continue to propagate until they arrive at a module with "1" hot syndrome input. Boundary modules do not send grow signals but they send pair_request signals when they receive grow signals; they also send pair signals when they receive pair_grant signals.

3.6 Implementation

3.6.1 SFQ Implementation of Greedy Decoding

SFQ is a magnetic pulse-based fabric with switching delay of *1ps* and energy consumption of 10^{-19}J per switching. In addition, availability of superconducting microstrip transmission lines in this technology makes it possible to transmit picosecond waves with half of speed of light and without dispersion or attenuation. The combination of these properties together with fast two-terminal Josephson junctions, makes this technology suitable for high speed processing of digital information [221, 128, 97, 203, 141]. SFQ logic families are divided into two groups: ac-biased and dc-biased; Reciprocal Quantum Logic (RQL) [97], and Adiabatic Quantum Flux Parametron (AQFP) [203] are in the first group, and Rapid Single Flux Quantum (RSFQ) [141], Energy-efficient RSFQ (ERSFQ) [128], and energy-efficient SFQ (eSFQ) [221] are examples of the second group. The dc-biased logic family with higher operation speed (as high as 770GHz for a T-Flip Flop (TFF) [44]) and less bias supply issues are more popular than ac-biased logic family.

Our algorithm requires modules to propagate signals one step at each cycle. One approach to implement our algorithm is to use synchronous elements such as flip-flops in decoder modules. However, standard CMOS style flip-flops are very expensive in SFQ logic (e.g., one D-Flip-Flop occupies 72.4× more area and consumes 117× more power compared to a 2-input AND gate). On the other hand, SFQ gates have a unique feature that we utilize to implement our algorithm without flip-flops. Unlike CMOS gates, most of the SFQ gates (expect for mergers, splitters, TFFs, and I/Os) require a clock signal to operate [172]. Thus,

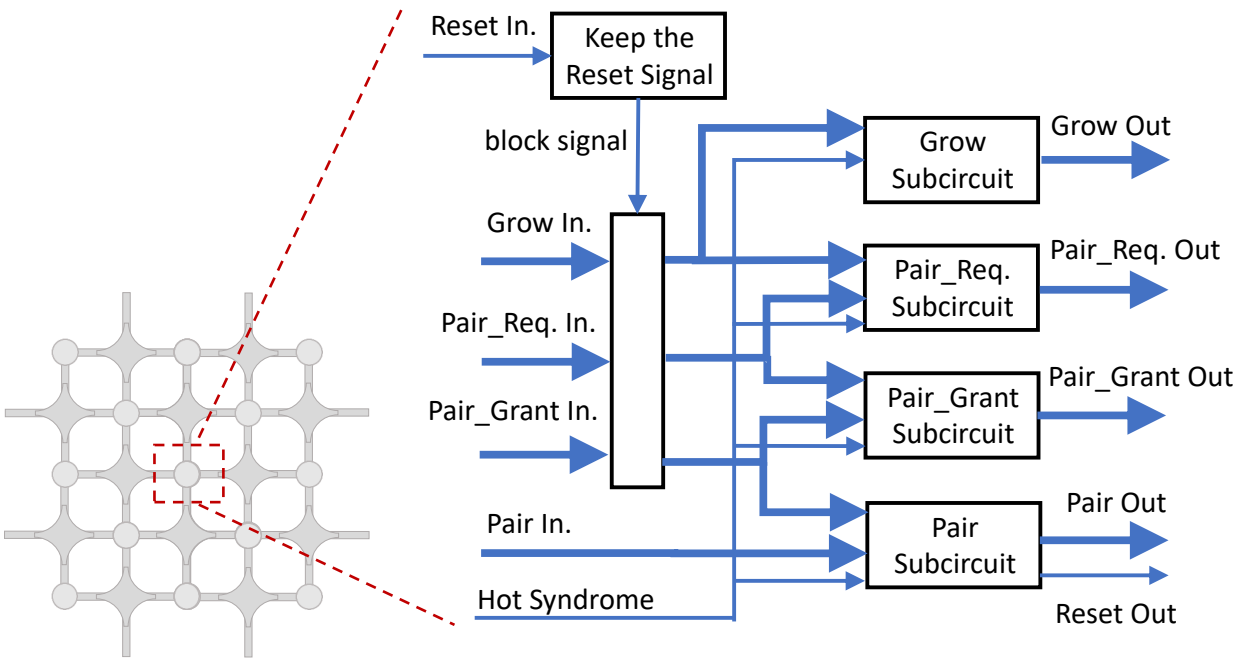


Figure 3.8: Overview of decoder module microarchitecture.

we do not need to have flip-flops and signals can propagate one SFQ gate at each cycle.

As described earlier, our decoder requires resetting the decoder modules each time two hot syndrome modules are paired. We have a global wire that passes through all the modules and is connected to each module using splitter gates. Thus, if we set the value of the global wire, all of the decoder modules receive the reset signal at the same time, as the splitter gates do not require clock signals to operate. If a module receives a reset signal, it blocks the module inputs using 2-input AND gates (one input is *module_input* and the other input is $R\bar{e}set$). In order to reset a decoder module completely, we need to block the module inputs for as many cycles as the depth of our SFQ-based decoder because the SFQ gates work with clock cycles and one level of gates is reset at each cycle. Thus, we use a simple circuit to keep the reset signal "1" for as many cycles as the circuit depth. In each module, we pass the reset signal that comes from the global wire to a set of m cascaded buffer gates where m is the circuit depth, and the module inputs are blocked if the reset signal that comes from the global wire is "1" or at least one of the buffers has "1" output.

3.6.2 Datapath and Subcircuit Design

Figure 3.8 shows an overview of our decoder module microarchitecture. Our decoder consists of five main subcircuits.

Grow Subcircuit: this subcircuit receives hot syndrome input and 4 grow inputs (from 4 different directions), and produces 4 grow output signals. Grow outputs are "1" if the hot syndrome input is "1" or if the module is passing a grow signal generated by another module.

Pair_Req Subcircuit: this subcircuit is responsible for setting the value of pair_request outputs which are "1" if two grow signals meet at an intermediate module or if the module is passing a pair_request signal that arrived at one of its input ports. The module does not pass the pair_request input signal if the hot syndrome input is "1"; in that case, the module generates a pair_grant signal instead.

Pair_Grant Subcircuit: this module determines the value of pair_grant outputs which are "1" if the module is a hot syndrome module and gives grant to a pair_request signal, or if the module is passing a pair_grant input signal to the adjacent module.

Pair Subcircuit: this subcircuit sets the value of pair outputs which are "1" if two pair_grant signals meet at an intermediate module or if a pair input signal is "1" and the hot syndrome input is not "1". If both the pair input and hot syndrome input are "1", the module does not pass the pair signal and instead generates a global reset signal that reset all of the decoder modules and also resets the hot syndrome input. Note that the reset signal resets everything except the subcircuit responsible for passing the pair signals because it is possible that the intermediate module does not have equal distance from the paired hot syndrome modules and we do not want to stop the propagation of all the pair signals in the system when the closer module receives a pair signal (while the farther module has not received a pair signal yet). The SFQ implementation of this subcircuit is shown in Figure 3.9.

Reset Subcircuit: this subcircuit is responsible to keep the reset signal "1" for as many cycles as the depth of our circuits. The depth is 5 in our circuits, thus reset subcircuit blocks grow, pair_req and pair_grant inputs for 5 cycles in order to reset the module.

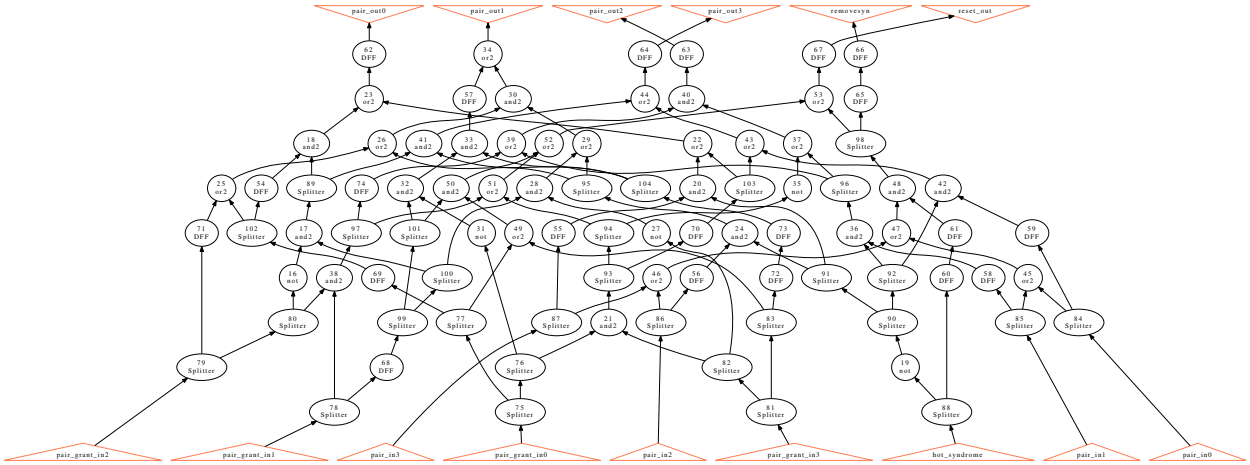
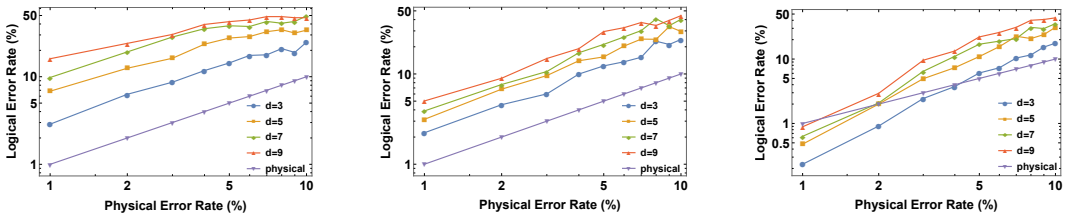


Figure 3.9: Pair subcircuit after SFQ specific optimizations and mapping. Triangular shapes at the bottom represent the primary inputs of the circuit and those at the top of the circuit show primary outputs. *DFF* is SFQ DRO DFF inserted for path balancing. Splitter (balanced) trees are also shown. Splitter is an asynchronous SFQ gate that receives a pulse at its input and after its intrinsic delay, it produces two almost identical output pulses. We insert splitters at the output of an SFQ gate (or a primary input) with more than one fanout.

3.7 Methodology

Simulation Techniques: In order to effectively benchmark the performance of a stabilizer quantum error correcting code, techniques must be used to simulate the action of the code over many cycles. This is referred to elsewhere in literature as *lifetime simulation* [213], or simply Monte Carlo benchmarking. We constructed a simulation environment that simulates the action of the stabilizer circuits. A cycle refers to one full iteration of the stabilizer circuit. At each step within the cycle, errors are stochastically injected into the qubits and propagated through the circuits. Ancillary qubits are measured, and the outcomes are reported in the error syndrome. This syndrome is then communicated directly to the decoder simulator, which returns the corresponding correction. The correction is applied and the surface is checked for a logical error. The ratio of the number of logical errors to the number of cycles run in simulation is used as the primary performance metric.

Evaluation Performance Metrics: In our evaluations, we use the stabilizer circuits as the primary benchmark. These circuits are replicated for every ancillary qubit present in



(a) Baseline design (b) Adding resets (c) Resets & boundaries

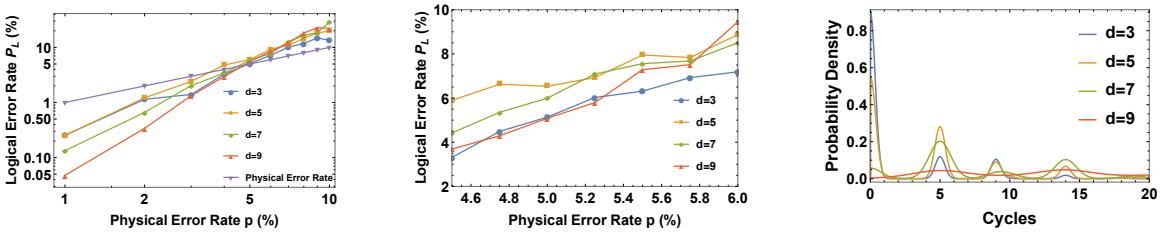
Figure 3.10: Logical error rate performance of each incremental design step. The addition of resets and boundaries each contribute heavily to the realization of pseudo-thresholds, and have a dramatic effect on reducing the minimum achievable logical error rates for each code distance.

a surface code lattice. Many different lattices are also analyzed, ranging in size from code distances 3 to 9.

As performance metrics, we focus on *accuracy thresholds* and *pseudo-thresholds*. The former is the physical error rate at which the code begins to suppress errors effectively across multiple code distances. Below this threshold, the logical error rate P_L decreases as the code distance d increases. Above threshold these relationships invert, and P_L grows with d due to decoder performance: the presence of many errors causes the decoding problem to become too complex. In many cases, this leads to corrections that complete what would have otherwise been short error chains, forming logical errors, a process that amplifies as code distances increase.

Pseudo-threshold refers to the performance of a single code distance, and is the physical error rate at which the logical error rate is equal to the physical rate, i.e. $P_L = p$. This can be (and often is) different across different code distances. Better error correcting codes will have higher pseudo-threshold values, as well as higher accuracy thresholds.

Error Models: The Monte Carlo simulation environment requires a model of the errors on the quantum system. We choose to focus on the *depolarizing channel* model [154, 74, 140, 206], parameterized by a single value p: Pauli X, Y, and Z errors occur on qubits with probability $p/3$. During simulation Pauli errors are sampled i.i.d for injection on each data qubit. We present analysis of a variation of the model, the *pure dephasing channel* [50, 51] comprised



(a) Final design (b) Zoomed in final design (c) Solution time statistics

Figure 3.11: Top row: Logical error rate performance of each incremental design step. The addition of resets and boundaries each contribute heavily to the realization of pseudo-thresholds, and have a dramatic effect on reducing the minimum achievable logical error rates for each code distance. Bottom row: Results for our final design, including support for reset, boundary, and equidistant mechanisms. (a) Error rate scaling for the proposed decoder. An accuracy threshold is evident at approximately 5% physical error rate, while pseudo-thresholds span the range from $\sim 3.5\% - 5\%$. (b) Logical error rates near the 5% physical error rate value. (c) Truncated unnormalized estimated probability distributions for the execution cycles required by each code distance in simulation. Window shows up to 20 cycles for comparison across code distances. Notice that while distances $3, 5, 7$ display peaks centered at $0, 5, 9$, and 14 cycles.

Cell	Area (μm^2)	JJ Count	Delay (ps)
AND2	3500	16	8.7
OR2	3500	14	6.0
XOR2	3500	18	6.3
NOT	3500	12	13.0
DRO DFF	3000	11	6.8

Table 3.2: The library of ERSFQ cells and corresponding characteristics used for synthesizing the circuit into SFQ hardware. Josephson Junction count is listed in the second column.

solely of Pauli Z errors occurring on qubits with probability p. The decoder will be operated symmetrically for both X and Z errors, allowing for simple extrapolation from these results.

Single Flux Quantum Circuit Synthesis: An ERSFQ library of cells is used in this paper to reduce the total power consumption (including the static and dynamic) of the surface code decoder as much as possible. Table 3.2 lists characteristics of this library. As seen, this library contains four logic gates including AND2, OR2, XOR2, and NOT, and it has a Destructive Read-Out D-Flip-Flop (DRO DFF) cell. Area of all logic cells are the same and it is equal to 3500 μm^2. However, area of the DRO DFF is less than the area of these gates (3000 μm^2). DRO DFFs are different from standard CMOS style flip-flips: they are

specially designed for SFQ circuits and are usually used for path balancing. In Table 3.2, the total number of Josephson junctions (as a measure of complexity and cost) used in designing each gate together with the intrinsic delay of each cell is reported.

The decoder circuit and its sub-circuits are synthesized by employing ERSFQ specific logic synthesis algorithms and tools [170, 172, 171, 169]. These algorithms are designed to reduce the complexity of the final synthesized and mapped circuits in terms of total area and Josephson junction count. This is achieved by reducing the required *path balancing* DFF count for realizing these circuits. Please note that for correct operation of dc-biased SFQ (including ERSFQ) circuits, these circuits should be fully path balanced; this means that in a Directed Acyclic Graph (DAG) that represents an SFQ circuit, length of any path from any primary input to any primary output in terms of the gate count should be the same. In most of the SFQ circuits this property does not hold in the beginning. Therefore, some path balancing DFFs should be inserted into shorter paths to maintain the full path balancing property. In the algorithms we employed for mapping these circuits, a dynamic programming approach is used to ensure minimization of the total number of DFFs to maintain the balancing property [170, 171]. In addition, a depth minimization algorithm together with path balancing is employed [172] to reduce the logical depth (length of the longest path from any primary input to any primary output in terms of the gate count) of the final mapped circuit. This helps to reduce the latency of the mapped SFQ circuit. As mentioned before, SFQ logic gates are pulsed-based, meaning that the presence of a pulse represents a logic-"1" and the absence of a pulse represents a logic-"0". Each gate is clocked, and as an example, the SFQ NOT gate behaves as follows. After the clock pulse arrives, when there are no input pulses, a pulse is generated at the output of the gate representing a "1". On the other hand, when there is an input pulse, no pulses are generated at the output, meaning a "0". Each pulse is a single quantum of magnetic flux ($\phi_0 = \frac{h}{2e} = 2.07\text{mV}\times\text{ps}$) [141]. To simulate the SFQ circuits for verifying their correct functionality, we use the Josephson simulator (JSIM) [52].

Circuit	Logical Depth	Latency (ps)	Area (μm^2)	Power (μW)
AND_GATE	1	8.7	3500	0.026
OR_GATE	1	6.0	3500	0.026
OR_GATE_7_INPUTS	3	18.0	33000	0.338
NOT_GATE	1	13.0	3500	0.026
Pair_Grant Subcircuit	5	115.0	293500	3.38
Pair Subcircuit	5	115.0	303500	3.53
Pair_Req./Grow Subcircuit	5	115.0	406500	4.75
Full_Circuit	6	168.0	1143000	13.44

Table 3.3: Experimental synthesis results for the SFQ Decoder. Shown are all gates utilized in the synthesis, as well as submodules that comprise the main circuit. Pair_Req. and Grow subcircuits have been combined into a single subcircuit.

Code Distance	Max	Average	Standard Deviation
3	3.86	0.29	0.59
5	9.58	0.74	1.13
7	14.7	2.06	2.05
9	19.8	3.93	3.21

Table 3.4: Decoder execution time in nanoseconds across each code distance studied and across all simulated error rates.

3.8 Evaluations

In this section we evaluate the performance of our proposed decoder design, both in terms of circuit characteristics including power, area, and latency, as well as error correction performance metrics of accuracy and pseudo-thresholds. We also analyze the execution time of our system, relying upon described operating assumptions and circuit synthesis results.

Threshold Evaluations: To gauge the performance of our design, we use the threshold metrics described in Section 3.7. Figure 4.10 (a) shows the central performance result, while the top row of Figure 4.10 shows the effect of all of the incremental design decisions on the overall performance. This evaluation simulates the performance of the decoder across a range of physical error rates. A pseudo-threshold range of between 3.5% and 5% is observed, and an accuracy threshold appears at approximately the 5% error rate. For code distance 5, the pseudo-threshold is below the accuracy threshold. This highlights the difference between these metrics – an error correcting protocol like the surface code can perform well even though particular code distances may still be amplifying the physical error rates (i.e. $P_L > p$). It is

Code Distance	3	5	7	9
c_2	0.650	0.429	0.306	0.323

Table 3.5: Empirical parameter estimation given a model of the form $P_L \approx c_1(p/p_{\text{th}})^{c_2 \cdot d}$. Shown are estimated c_2 parameter values.

important to consider both types of thresholds when evaluating decoder performance.

An interesting behavior is observed for code distance $d = 3$. This lattice performs at or surpassing the performance of all other lattices from the 3% physical error rate and above. Below this point, the lattice begins to taper off, and ultimately it converges with the distance 5 lattice. Boundary conditions were highly prioritized in our design, causing this effect. In particular, the decoder is designed such that error chains that terminate at the boundaries are more likely to be correctly identified than other patterns. This choice was made as smaller lattices are more dominated by these edge effects than larger lattices. The smallest lattice in our simulations shows this anomalous behavior, as it contains a disproportionate amount of boundary patterns. In larger lattices, syndromes are less likely to terminate in boundaries, reducing this effect.

Figure 4.10 (b) highlights the desired threshold behavior. Examining the 6% error rate, code distance 9 is outperformed by code distance 7. Moving to the lowest physical error rate in the window, we find that the lattices perform in the order $d = 9$, $d = 3$, $d = 7$, and $d = 5$, ordered from lowest to highest logical error rate. Barring the anomalous $d = 3$ behavior described above, this is accuracy threshold behavior indicative of successful error correction performance.

Performance Analysis: To quantify the approximation factor of our design we compare the performance to that of an ideal decoder by fitting to an exponential analytical model. The achievable error rates by the surface code ideally can be described by $P_L \approx 0.03(p/p_{\text{th}})^d$ [74] when a minimum weight matching decoder is used in software. This model is valid for the error models we consider, as [74] uses and fits "class-0" Pauli errors in the same fashion. Using a model of the form $P_L \approx c_1(p/p_{\text{th}})^{c_2 \cdot d}$, we fit values of c_1, c_2 for each code distance

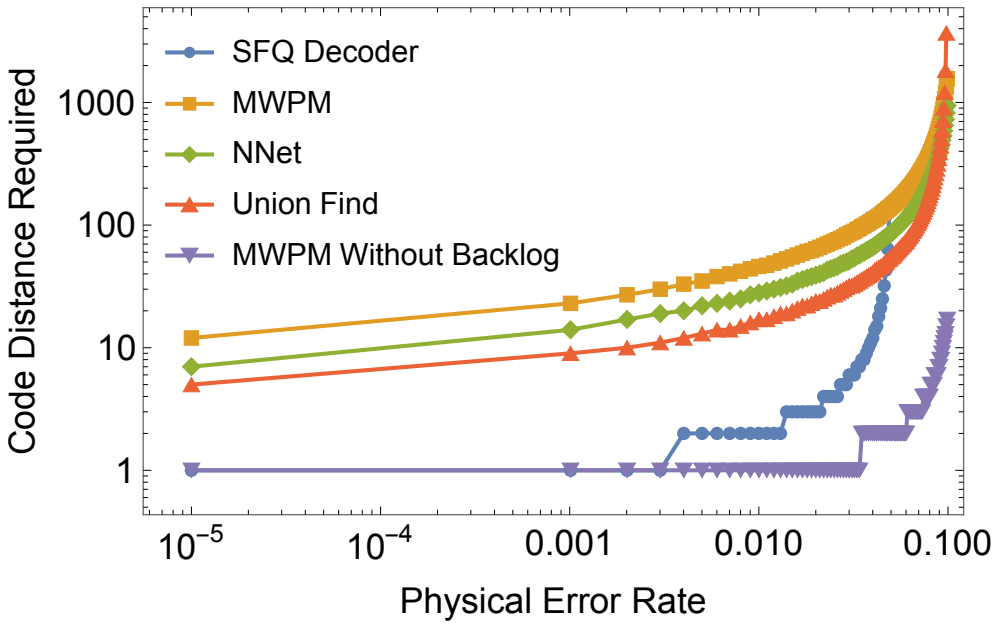


Figure 3.12: Comparison of required code distances of different decoders to execute an algorithm consisting of 100 T-gates. Compared are the SFQ Decoder, minimum weight perfect matching decoder (MWPM) [74], neural network decoder [14], union find decoder [50], and a theoretical MWPM decoder with no backlog. across both code distances and physical error rates.

at physical error rates below accuracy threshold, and collect c_2 values in Table 3.5. C_2 coefficients describe the *effective code distance* for our system, and capture the approximation factor we introduce. For code distances 3 and 5, we find that the approximate decoder is roughly 65% and 43% of the optimal distance respectively. This is the trade-off made by our system in order to fit the timing and physical footprint requirements of the system.

Notice that this accuracy tradeoff results a net resource reduction for our design over other proposed designs as shown in Figure 3.12. The data backlog imposes delays into the system that decrease the logical accuracy of any decoder that incurs this backlog. As the backlog builds up, the number of required syndrome detection cycles builds up as well, resulting in a new effective logical error rate as one logical gate now requires many more syndrome detection cycles to occur. The SFQ decoder pays an accuracy price for speed, but when the backlog is taken into consideration this tradeoff results in a significant performance gain over

alternative designs.

Synthesis Results and Circuit Characterization: Table 3.3 shows experimental results for the surface code decoder circuit presented in this paper using the aforementioned ERSFQ library of cells described in Section 3.7. The full circuit demonstrates a cycle latency of 168 ps, and an area and power footprint of 1.143 mm^2 and 13.44 μW, respectively. The full decoder is comprised of a mesh of these circuit modules, requiring a single module per individual qubit. This means that for systems of code distance 9 comprised of 289 qubits, the decoder required will be of size 330.33 mm^2 and will dissipate 3.88 mW of power. Typical dilution refrigerators are capable of cooling up to $1-2$ Watts of power in the 4-Kelvin temperature region [105], enabling the co-location of a decoder mesh of size 87×87, which would protect a single qubit of code distance $d = 44$, or 100 qubits of code distance $d = 5$. These values are estimations given modern day SFQ and cryogenic dilution refrigerator technology, much of which is subject to change in the future.

Execution Time Evaluation: The most important characteristic that the SFQ decoder aims to optimize is real-time execution speed. Previous works have described the syndrome generation time to be between $160-800$ ns for superconducting devices that we are focusing on in this study. [84, 205].

In practice the time to solution is much lower than the upper bound of $O(n)$ on the greedy algorithm. Table 3.4 contains the empirically observed statistics of our decoder operation. The maximum cycles to solution is well approximated by a linear scaling with a leading coefficient of ~ 15.75. Statistical distributions describing the required cycles to solution for each code distance are shown in Figure 4.10 (c).

Comparison to existing approximation techniques: Trading the accuracy for decoding speed has been utilized in prior work. Union-find [50] achieves a significant speed-up over the minimum weight perfect matching algorithm, while the accuracy threshold decreases by only 0.4%. Despite this, the union-find decoding time is still longer than the syndrome generation time ($> 2X$ longer) thus exposing it to the exponential latency overhead caused

by the data backlog. In contrast to prior approximation techniques, decoding time in our design is faster than syndrome generation time and thus it does not incur exponential latency overhead, enabling a practical implementation of error-correcting quantum machines.

Effect on SQV: The net effect of our design is to expand the SQV achievable by near-term machines. An example of this is a small 100 physical qubit system in which 10^3 gates are performed per qubit, a machine that is conjectured will exist in the near future [25] and shown in red in Figure 3.1. By utilizing the scaling equation described in Section 3.8, we see that a homogeneous system of 78 logical qubits each of code distance 3 is capable of performing $\sim 4.36\times10^6$ gates per qubit. This expands SQV from $100\times10^3 \rightarrow 78\times(4.36\times10^6) \approx 3.4\times10^8$, increasing by a factor of 3402. This can be pushed farther by going to the small qubit count limit, constructing a machine of 40 logical qubits each of code distance 5 with logical error rate 8.96×10^{-10}, yielding SQV of $\approx 1.12 \times 10^9$, an increase of $11,163$. These effects are captured in Figure 3.1. Not all applications benefit from these expansions in the same fashion, but our techniques allow for machines to be used in ways that are tailored to individual applications, and enable much more computation to be performed on the same machine.

3.9 Conclusion

In the design of near-term quantum computers, it is vital to enable the machines to perform as much computation as possible. By taking inspiration from quantum error correction, we have designed an "Approximate Quantum Error Correction" error mitigation technique that expands the "Simple Quantum Volume" of near-term machines by factors between 3,402 and 11,163. Our design focuses on the construction of an approximate surface code decoder that can rapidly decode error syndrome, at the cost of accuracy.

Using SFQ synthesis tools, we show that the area and power are within the typical cryogenic cooling system budget. In addition, our accelerator is based on a modular, scalable architecture that uses one decoder module per each qubit. Most importantly, our decoder constructs solutions in real-time, requiring a maximum of ∼20 ns to compute the solution in

simulation. This allows our decoding accelerator to achieve 10x smaller code distance when compared to offline decoders when accounting for decoding backlog. Thus, it is a technique that can effectively boost the Simple Quantum Volume of near-term machines.

CHAPTER 4

MAGIC-STATE FUNCTIONAL UNITS: MAPPING AND SCHEDULING MULTI-LEVEL DISTILLATION CIRCUITS FOR FAULT-TOLERANT QUANTUM ARCHITECTURES

Quantum computers have recently made great strides and are on a long-term path towards useful fault-tolerant computation. A dominant overhead in fault-tolerant quantum computation is the production of high-fidelity encoded qubits, called *magic states*, which enable reliable error-corrected computation. We present the first detailed designs of hardware functional units that implement space-time optimized *magic-state factories* for surface code error-corrected machines.

Interactions among distant qubits require *surface code braids* (physical pathways on chip) which must be routed. Magic-state factories are circuits comprised of a complex set of braids that is more difficult to route than quantum circuits considered in previous work [112]. This paper explores the impact of scheduling techniques, such as gate reordering and qubit renaming, and we propose two novel mapping techniques: braid repulsion and dipole moment braid rotation. We combine these techniques with graph partitioning and community detection algorithms, and further introduce a stitching algorithm for mapping subgraphs onto a physical machine. Our results show a factor of 5.64 reduction in space-time volume compared to the best-known previous designs for magic-state factories.

4.1 Introduction

Quantum computers of intermediate scale are now becoming a reality. While recent efforts have focused on building Noisy Intermediate-Scale Quantum (NISQ) computers without error correction, the long-term goal is to build large-scale fault-tolerant machines [178]. In fault-tolerant machines, typical quantum workloads will be dominated by error correction [205]. On machines implementing *surface code* error correction, fault-tolerant operations

known as *magic-state distillation* will make up the majority of the overhead. The problem of achieving effective magic-state distillation is two-fold: 1) useful quantum applications are dominated by magic-state distillation, and 2) their support is extremely expensive in both physical area and latency overhead. The innovations in this paper address the largest obstacle facing large-scale quantum computation.

Magic-state distillation requires the preparation (i.e. *distillation*) of high-fidelity logical qubits in a particular state, which can enable the execution of fault-tolerant instructions. These states require expensive, iterative refinement in order to maintain the reliability of the entire device.

This work proposes optimizations for the architectural functional units (i.e. "factories") to generate magic states. Using a realistic resource overhead model, we introduce optimization techniques that exploit both instruction level scheduling as well as physical qubit mapping algorithms. Our approach analyzes and optimizes, for the first time, fully mapped and scheduled instances of resource state generation units known as multilevel block-code state-distillation circuits. We develop novel technology-independent heuristics based upon physical dipole-moment simulation to guide annealing algorithms aiming to discover optimized qubit register mappings. We use these along with a new combination of conventional compiler methods to exploit structure in the distillation circuitry. Together, these techniques reduce resource overhead (space-time volume) by 5.64x. We make use of a novel software toolchain that performs end-to-end synthesis of quantum programs from high level expression to an optimized schedule of assembly gate sequences, followed by intelligent physical qubit register allocation, and surface code simulation.

Our techniques are based on analysis of circuit interaction graphs, where nodes represent qubits and edges represent operations between the endpoints. We show that a combination of graph partitioning-based mapping procedures and dipole-moment driven annealing techniques work well on structured surface code circuits. State distillation circuits can be subdivided cleanly into sets of disjoint planar subgraphs. We find that each of these planar subgraphs can

be mapped nearly optimally. The higher level structure of the distillation circuits introduces non-trivial permutation steps between circuit subdivisions as well. We present an algorithm that combines optimized subgraph mappings with a force-directed annealing technique that optimizes the transition between the levels of the circuit. This technique is compared to conventional, global methods that optimize for specific characteristics of the interaction graph. The planar graph extraction and "stitching" technique outperforms global methods.

In summary, this paper makes the following contributions:

- We study the characteristics of two-qubit interactions in surface code error corrected machines, and show strong correlation between circuit latency and the number of edge crossings in the circuit interaction graph.
- We use this information to develop a heuristic inspired by simulation of molecular dipoles, and show that this can be used to generate low-latency qubit mappings by reducing edge crossings.
- We exploit the structure of the state distillation circuits to optimize individual rounds of distillation separately, and combine these rounds with optimized permutation networks to generate the lowest resource-overhead implementation of distillation units to date.

The rest of the paper is organized as follows: Section 5.2 describes quantum computation, surface code error correction, and magic state distillation in more detail. Section 7.2 describes related work that aims to optimize state distillation. Section 4.4 clarifies and summarizes the techniques we use to result in efficient factory circuits. Sections 4.5 and 4.6 specifically describe the scheduling properties of these circuits and mapping techniques along with heuristics utilized to optimize the procedures. Section 4.7 describes in greater detail the fully optimized algorithm for achieving efficient factory circuits. Section 7.7 describes the results we obtain. Finally, Sections 5.10 and IV discuss future work and conclude.

4.2 Background

4.2.1 Basics of Quantum Computation

Quantum computation involves the manipulation of fragile quantum states by operating on quantum bits (qubits). Each qubit is capable of existing in a superposition of two logical states $|0\rangle$ and $|1\rangle$ written as a linear combination $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, for complex coefficients α, β such that $|\alpha|^2 + |\beta|^2 = 1$. Upon measurement, the qubit state "collapses" to either $|0\rangle$ or $|1\rangle$. $|\alpha|^2$ and $|\beta|^2$ correspond to the probability of obtaining a $|0\rangle$ or $|1\rangle$ respectively. It is sometimes useful to visualize the state of a single qubit as a vector on the Bloch sphere [26, 154], because we can reinterpret the state $|\psi\rangle$ in its spherical coordinates as $|\psi\rangle = \cos(\theta/2)\,|0\rangle + \exp(i\phi)\sin(\theta/2)\,|1\rangle$. Any operations (quantum gates) performed on a single qubit can thus be regarded as rotations by some angle φ along some axis $\hat{n}$, denoted as $R_{\hat{n}}(\varphi)$. This work focuses on the phase gate ($S \equiv R_z(\pi/2)$), the T gate ($T \equiv R_z(\pi/4)$), and the most common two-qubit gate called controlled-NOT (CNOT) gate.

Quantum computing systems are commonly characterized by the maximum supportable *space-time volume* of a computation. This is the product of the number of qubits in the system with the number of operations (i.e. timesteps) that can be performed on the system reliably [25]. Reliable machines can be built in a variety of ways, each of which may result in a different combination of physical qubit count and computation time. To capture this, the space time volume of a computation is a useful metric by which computations and architectural solutions can be compared.

4.2.2 Surface Code Error Correction

Quantum states decohere over time which can result in performance loss and failure to produce the correct output. In order to maintain the advantage that quantum computation offers while balancing the fragility of quantum states, quantum error correction codes (QECC) are utilized to protect quantum states undergoing a computation. One of the most prominent

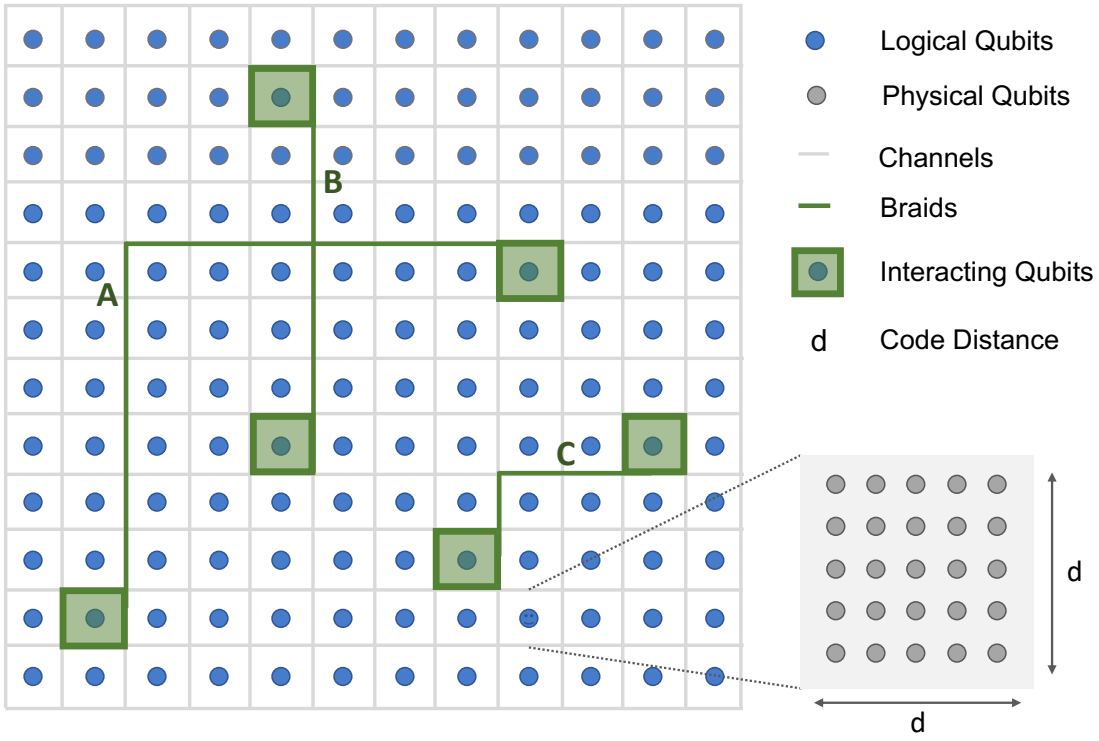


Figure 4.1: An array of (blue) logical qubits in a quantum processor. Highlighted lines indicate *braids* implementing two qubit interactions. These braids must exist spatially and temporally as pathways between qubits. This introduces communication congestion that depends upon specific architectural designs. Braid A and B are *crossing* braids, which cannot be executed simultaneously, while braid C is isolated and free to execute. Bottom-right inset represents a single logical qubit tile comprised of approximately d^2 physical qubit.

quantum error correcting codes today is the *surface code* [53, 69, 117]. These codes are a family of quantum error correcting codes that encode logical qubit states into the collective state of a lattice of physical qubits utilizing only nearest neighbor interactions between qubits designated as *data* and *ancilla* qubits. For a comprehensive introduction see an excellent tutorial in [69].

An important parameter of the surface code is the *code distance* d. The surface code can protect a logical state up to a specific fidelity P_L, which scales exponentially with d. More precisely, $P_L \sim d(100\epsilon_{in})^{\frac{d+1}{2}}$, where ϵ_{in} is the underlying physical error rate of a system [69]. Each logical qubit is made up of approximately d^2 physical qubits, as Fig. 4.1 shows.

4.2.3 CNOT Braiding

A *braid*, as illustrated in Fig. 4.1, is a path in the surface code lattice, or an area where the error correction mechanisms have been temporarily disabled and which no other operations are allowed to use. In other words, braids are not allowed to cross. In braiding, a logical qubit is entangled with another if the pathway encloses both qubits, where enclosing means extending a pathway from source qubit to target qubit and then contracting back via a (possibly different) pathway. These paths can extend up to arbitrary length in constant time, by disabling all area covered by the path in the same cycle.

4.2.4 T Gates in Quantum Algorithms

S and T rotation gates are important operations in many useful quantum algorithms, and their error-corrected execution requires magic state resources. When the number of T gates in an application is low, the circuit is in fact able to be efficiently simulated classically [33]. T gates have been shown to comprise between 25% and 30% of the instruction stream of useful quantum applications [205]. Others claim even higher percentages for specific application sets, of between 40% and 47% [111].

For an estimate of the total number of required T gates in these applications, take as an example the algorithm to estimate the molecular ground state energy of the molecule Fe_2S_2. It requires approximately 10^4 iteration steps for "sufficient" accuracy, each comprised of 7.4×10^6 rotations [223]. Each of these controlled rotations can be decomposed to sufficient accuracy using approximately 50 T gates per rotation [131]. All of this combines to yield a total number of T gates of order 10^{12}. As a result, it is crucial to optimize for the resource overhead required by the execution of T gates at this scale to ensure the successful execution of many important quantum algorithms.

4.2.5 *T Magic States*

T and S gates, while necessary to perform universal quantum computation on the surface code, are costly to implement under surface code. The number of T gates present in an algorithm is the most common metric for assessing how difficult the algorithm is to execute [194, 8]. To achieve fault-tolerance, an ancillary logical qubit must be first prepared in a special state, known as the *magic state* [35]. A distilled magic-state qubit is interacted with the data to achieve the T gate operation, via a probabilistic *injection* circuit involving 2 CNOT braids in expectation. For simplicity, because of their rotation angle relationship, we assume all S gates will be decomposed into two T gates.

These ancillary quantum states are called magic states because they enable universal quantum computation. Magic states can be prepared using Clifford quantum operations [35]. Since the task of preparing these states is a repetitive process, it has been proposed that an efficient design would dedicate specialized regions of the architecture to their preparation [200, 117]. These *magic state factories* are responsible for creating a steady supply of low-error magic states. The error in each produced state is minimized through a process called *distillation* [34].

4.2.6 *Bravyi-Haah Distillation Protocol*

Distillation protocols are circuits that accept as input a number of potentially faulty raw magic states, use some ancillary qubits, and output a smaller number of higher fidelity magic states. The input-output ratio, denoted as $n \to k$, assesses the efficiency of a protocol. This work focuses on a popular, low-overhead distillation protocol known as the Bravyi-Haah distillation protocol [34, 72].

To produce k magic states, Bravyi-Haah state distillation circuits take as input $3k + 8$ low-fidelity states, use $k + 5$ ancillary qubits, and k additional qubits for higher-fidelity output magic states, thus denoted as the $3k + 8 \to k$ protocol. The total number of qubits involved in each of such circuit is then $5k + 13$, which defines the area cost of the circuit module.

The intuition behind the protocol is to "make good magic states out of bad ones". Given a number of low-fidelity states, the protocol uses a syndrome measurement technique to verify quality, and discards states that are bad. Then, the circuit will convert the subset of good states into a single qubit state. The output magic states will have a suppression of error, only if the filtering and conversion follows a particular pattern. This is specified by the parity-check matrix in the protocol. Notably, if the input (injected) states are characterized by error rate ϵ_{inject}, the output state fidelity is improved with this procedure to $(1+3k)\epsilon_{\text{inject}}^2$. Due to the filtering step, the success probability of the protocol is, to first order, given by $1-(8+3k)\epsilon_{\text{inject}}+\cdots$.

4.2.7 Block Codes

Magic state distillation circuits operate iteratively and hierarchically. Often one iteration of the distillation procedure is not enough to achieve the desired logical error rate for a given program. In these cases, squaring the input error rate will not achieve the required logical error rate to execute the program. Instead, we can *recursively* apply the Bravyi-Haah circuit a number ℓ times, in order to achieve the desired error rate [116]. Constructing high fidelity states in this fashion is known as *block code* state distillation.

As Fig. 5.2 illustrates, ℓ level implementations of this procedure can be constructed recursively that support k^ℓ total output states at fidelity $\sim \epsilon_{\text{inject}}^{2^\ell}$, while requiring $(3k+8)^\ell$ input states.

The structure of the multi-level block code distillation protocol requires that each module takes in at most one state from each module from the previous round. This is because the magic states produced by one module may have correlated errors. So in order to avoid having correlated error in the inputs to the next round, each magic state from one module must be fed into a different module.

At the end of each individual module, error checking is performed. If the ancillary states show correct measurement results, the procedure was successful. Additional quality checks

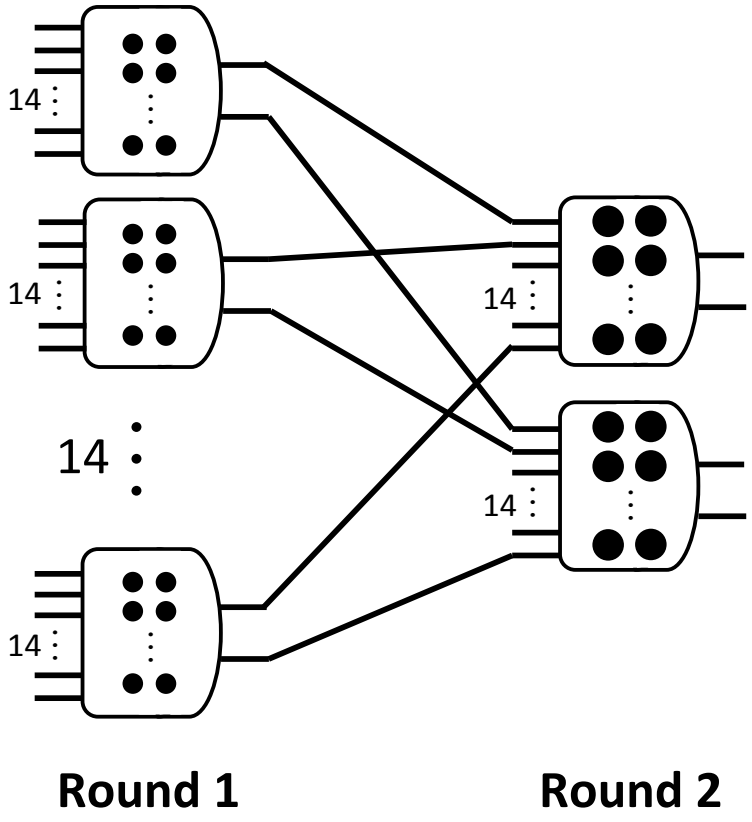


Figure 4.2: The recursive structure of the block code protocol. Each block represents a circuit for Bravyi-Haah $(3k+8) \to k$ protocol. Lines indicate the magic state qubits being distilled, and dots indicates the extra $k+5$ ancillary qubits used, totaling to $5k+13$. This figure shows an example of 2-level block code with $k = 2$. So this protocol takes as input $(3k+8)^2 = 14^2$ states, and outputs $k^2 = 4$ states with higher fidelity. The qubits (dots) in round 2 are drawn at bigger size, indicating the larger code distance d required to encode the logical qubits, since they have lower error rate than in the previous round [156].

were proposed by [156], which inserts a checkpoint at the end of each level of the factory. This checkpoint discards groups of modules when it detects failure within any of the modules in a group.

Within any particular round r of an ℓ-level magic state factory, the number of physical qubits required to implement that round defines the *space* occupied by the factory during round r. Because the output error rates drop each round, the required code distance increases accordingly. By the "balanced investment" technique shown in [156], each logical qubit in round r is constructed using $\sim d_r^2$ physical qubits, where each d_r varies with each round. The

idea is to use a smaller code distance to encode a logical qubit in earlier rounds of distillation to minimize area overhead.

In general, any particular round r may require several groups of Bravyi-Haah circuit modules. We denote the number of groups and number of modules per group as g_r and m_r respectively. The number of physical qubits q_r required to implement that round scales exponentially with $\ell - r$ as: $q_r = m_r^{r-1} g_r^{\ell-r}(5k + 13)d_r^2$. This exponential scaling plays a key role in our mapping techniques.

4.3 Related Work

Other work has focused primarily on optimizing the efficiency of the magic state distillation protocol. The original proposal [36] considered a procedure by which 15 raw input states would be consumed to produce a single higher fidelity output state. Later works [34, 116, 93] each explore different realizations of procedures that distill high fidelity magic states, with each procedure optimizing for asymptotic output rate and increasing this rate from the original proposal. These approaches tend to omit overheads related to actual circuit implementations.

Several prior works [72, 156] have attempted to reduce the circuit depth of an explicit implementation of the Bravyi-Haah distillation circuit, as well as perform a resource estimate by considering the rates at which these factories fail. Specifically, the work [72] by Fowler et al. is used as a baseline in this paper.

Additionally, several efforts have been made to build compilers and tools to be more precise about resource estimation quantification in topological quantum error corrected systems [163, 165, 164]. These techniques have resulted in tools that are used to compile and schedule arbitrary quantum circuits to topological assembly, and topological braid compaction techniques are used to reduce circuit depth expansions.

Systems level analysis has been performed by two related projects [111, 205], in which the former optimizes the structure of early distillation protocols, and the latter proposes a micro-architectural accelerator to handle large amounts of error correction instructions that

exist in fault tolerant machines.

Surface code braid scheduling costs were analyzed in [112] using an end-to-end toolflow. The work focused on the resource impact of the choice of different implementation styles of surface code logical qubits. That work provides a toolchain upon which we have built in order to optimize scheduling and mapping procedures, as well as perform circuit simulations.

Our work introduces the complexity of braid scheduling into the analysis of the structure of the leading state distillation procedures in an attempt to concretize the procedures into real space and time resource costs. The new annealing heuristics (e.g. dipole-moments) developed specifically for this purpose also generalize well to any circuit executing on a fault tolerant machine that uses braiding to perform two-qubit gates.

4.4 Our Approach

In order to minimize the space-time volume spent on multilevel magic state distillation, our approach takes advantage of the unique characteristics of the state distillation circuitry. We decompose the problem into two aspects – scheduling gate operations and mapping qubits into 2-D mesh. These two are intertwined, as the schedule determines what pairs of qubits *need* to interact, and mapping influences which subset of them *can* interact in the same cycle. An important tool used to perform these optimizations is the program interaction graph, from which circuit structure can be extracted. In particular, we combine the fact that these state distillation circuits are characterized by natural subdivisions between levels of the factory, with the ability of graph partitioning embedding techniques to nearly-optimally map small planar subgraphs of the program. We exploit this information to design a procedure that decomposes state distillation into components that are independently optimized. The flow chart of the procedure is illustrated in Fig. 4.3.

Levels of the factory are joined by a specific permutation of the output states exiting from previous rounds of distillation, which appears to impose significant overhead on the whole distillation process. To address this, a force-directed annealing algorithm is used in

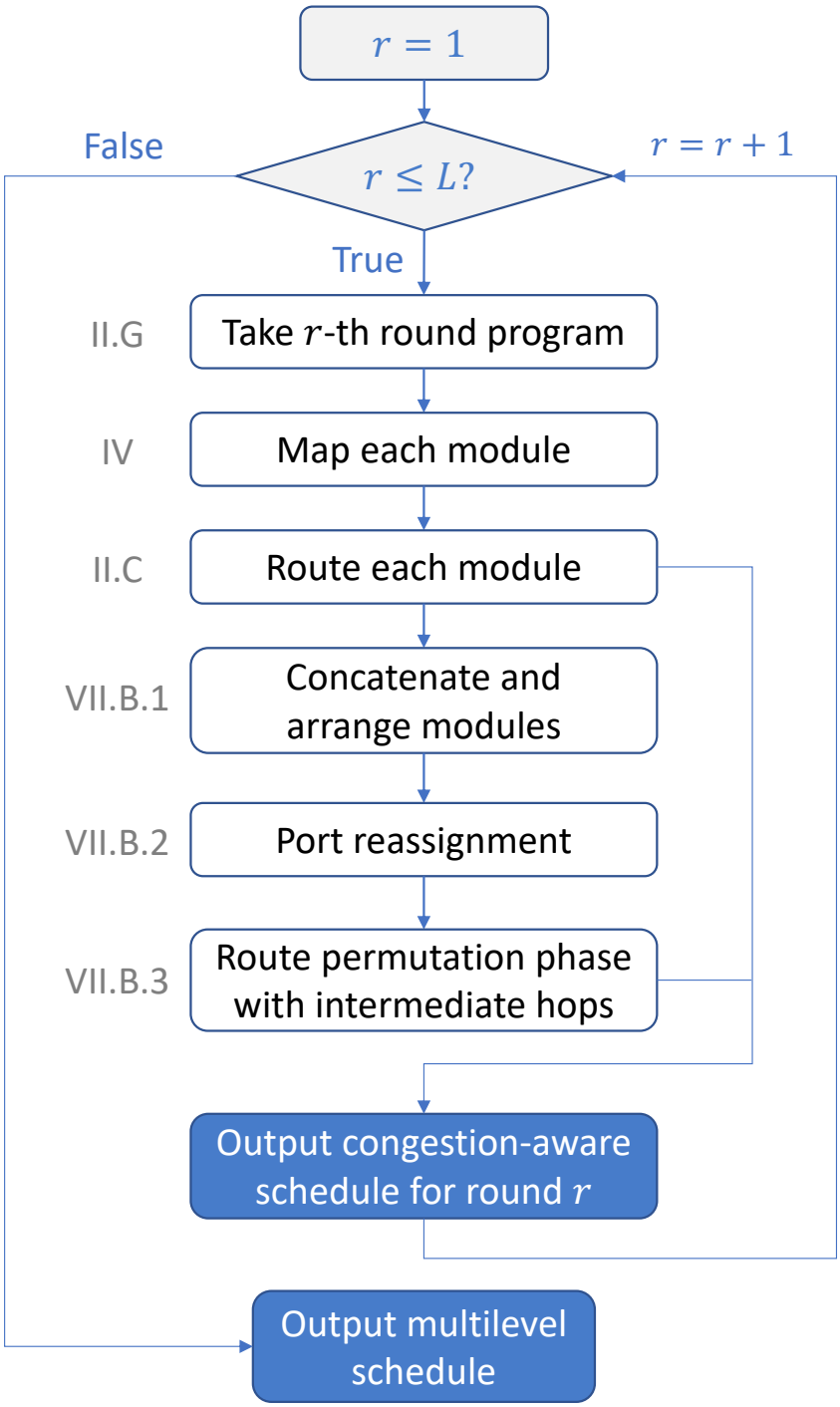
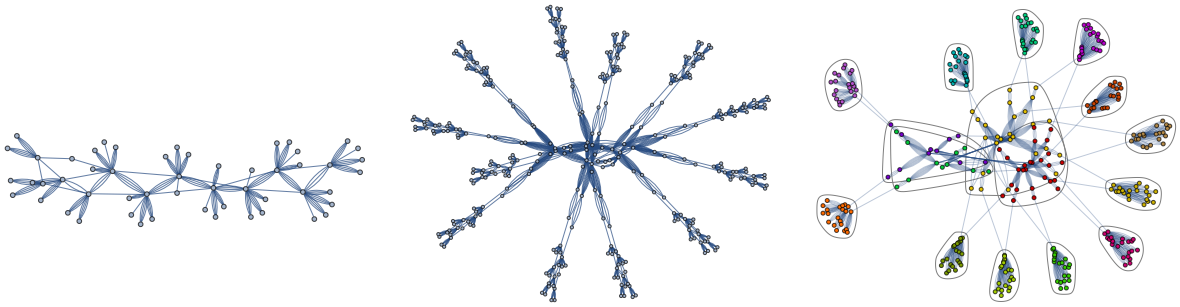


Figure 4.3: Flow chart for the overall approach, along with the section numbers corresponding to each component.



(a) Planar interaction graph of a capacity 8, single level factory

(b) Non-planar interaction graph of a capacity 4, two level factory

(c) Multi-level factory interaction graph with community structure

Figure 4.4: Interaction graphs of single and two level factories, and community structure of a capacity 4 two level factory. Each vertex represents a distinct logical qubit in the application, and each line represents a required two (or more) qubit operation. (a) shows that the single level distillation circuit has planar interaction graph, so mapping vertices to physical location in quantum processor is relatively simple. Each level in a multi-level factories like (b) have these planar substructures, but the permutation edges between rounds destroy the planarity of the two-level ineraction graph. (c) shows that we can leverage the planarity within each level by exploring community structure of the interaction graph, as shown in Section 4.6.

conjunction with ideas inspired by Valiant intermediate-destination routing for permutation networks [136] to reduce the latency of these permutation steps between block code levels.

The next few sections describe the scheduling and mapping optimizations decoupled from one another, in order to show the specific strengths and weaknesses of each. Section 4.7 then synthesizes these optimizations into a single procedure.

4.5 Scheduling

This section describes the impact of instruction level optimizations: gate scheduling and qubit reuse. A *schedule* of a quantum program is a sequence of gate operations on logical qubits. The sequence ordering defines *data dependencies* between gates, where a gate g_1 depends on g_0 if they share a logical qubit and g_1 appears later in the schedule sequence than g_0.

4.5.1 Gate Scheduling

The impact of gate scheduling can be quite significant in quantum circuits, and many algorithm implementations rely upon the execution of gates in parallel in order to achieve substantial algorithmic speedup. Gate scheduling in quantum algorithms differs from classical instruction scheduling, as gate commutativity introduces another degree of freedom for schedulers to consider. Compared to the field of classical instruction scheduling, quantum gate scheduling has been relatively understudied, with only few systematic approaches being proposed that incorporate these new constraints [85].

In exploring these effects applied to Bravyi-Haah state distillation circuits, we find that these optimizations are limited in their effectiveness. While intuitively the modularity of the block code construction would allow for early execution of gates arising in late rounds of the distillation procedure, the *checkpoints* required to implement module checking as described in section 5.2.5 limit the magnitude of gate mobility.

The structure of the block code circuitry only allows for a small constant number of gates to by executed early, outside of the rounds from which they originate. Because of this, the maximum critical path extension by the introduction of a *barrier* preventing gate mobility outside of the originating round is equal to this small constant multiplied by the number of block code iterations. The benefit of inserting a barrier at the end of each round is to create clean divisions between the rounds. As Fig. 4.4 shows, the interaction graph for a single round is a planar graph, while this planarity is destroyed as rounds are added. Barriers expose this planarity, making the circuit easier to map. Barriers in these circuits can be inserted by adding a multi-target CNOT operation into the schedule, controlled by an ancilla qubit initialized into a logical $|0\rangle$ state, and targeting all of the qubits that the schedule wishes to constrain.

Additionally, gate scheduling order has significant impacts on network congestion. Scheduling these small constant number of gates early therefore runs the risk of causing congestion with previous round activity. This can in fact extend the circuit latency, even though the

circuit has executed gates earlier in the schedule.

Overall, the insertion of a barrier appears to not significantly alter the schedule of circuit gates. It does, however, change the interaction between the schedule and a particular physical qubit mapping. This relationship will be explored in more detail in Section 4.7.

4.5.2 Qubit Reuse

We show in this section that an important schedule characteristic of the block protocol to leverage is the hierarchical structure of the distillation circuit. Between two rounds of the procedure, all ancillary qubits will be measured for error checking at the end of the previous round, and reinitialized at the beginning of the next round. This type of data qubit sharing (which we call "sharing-after-measurement") is a *false dependency*, because they can be resolved by qubit renaming. Now this naturally leads to the question: (how) should we reuse the qubits between multiple rounds?

The first approach we explore is to prevent any false sharing of the qubits, at the cost of larger area, by always allocating new data qubits for different rounds. This removes all dependencies due to ancillary qubits, leaving only true dependencies on qubits generated in the previous round. This minimizes execution time at the cost of extra qubits (and space).

The second approach is to strategically choose which qubits from the previous round to be reused for the next. This approach directly reduces the area needed for the entire factory, at the cost of introducing false dependencies.

In order to make intelligent decisions on which set of ancillary qubits to reuse, it requires us to have information about the topological mapping of the qubits, since mapping and reuse decisions together significantly influence the congestion overhead of the circuit. We will discuss the subtleties of the tradeoff in more detail later in Section 4.7.

```
// Bravyi-Haah Distillation Circuit with K=8, L=1
#define K 8

module tail(qbit* raw_states, qbit* anc, qbit* out) {
  for (int i = 0; i < K; i++) {
    CNOT ( out[i] , anc[5 + i] );
    injectT ( raw_states[2 * i + 8 + i] , anc[5 + i] );
    CNOT ( anc[5 + i] , anc[4 + i] );
    CNOT ( anc[3 + i] , anc[5 + i] );
    CNOT ( anc[4 + i] , anc[3 + i] );
  }
}

module BravyiHaahModule(qbit* raw_states, qbit* anc, qbit* out) {
  H ( anc[0] );
  H ( anc[1] );
  H ( anc[2] );
  for (int i = 0; i < K; i++)
      H ( out[i] );
  CNOT ( anc[1] , anc[3] );
  CNOT ( anc[2] , anc[4] );
  CXX ( anc[0] , anc , K );
  tail( raw_states , anc , out );
  for (int i = 1; i < K + 5; i++)
    injectT(raw_states[2 * i - 2], anc[i]);
  CXX ( anc[0] , anc , K + 4 );
  for (int i = 1; i < K + 5; i++)
    injectTdag(raw_states[2 * i - 1], anc[i]);
  MeasX ( anc );
}

/* Single-level circuit requires a single module.
 * Multi-level circuits would require more modules
 * and barriers in this function. */
module block_code(qbit* raw, qbit* out, qbit* anc) {
  BravyiHaahModule( raw , anc , out );
}

module main (  ) {
  qbit raw_states[3 * K + 8];
  qbit out[K];
  qbit anc[K + 5];
  block_code( raw_states , out , anc );
}
```

Figure 4.5: Example implementation [72, 1] of a single-level Bravyi-Haah distillation circuit generating $K = 8$ output magic states, in Scaffold language [113]. The corresponding interaction graph is illustrated in Fig. 4.4a. `injectT` and `injectTdag` implement the probabilistic magic state injection described in 5.2.2. `CXX` implements a single-control multi-target CNOT gate.

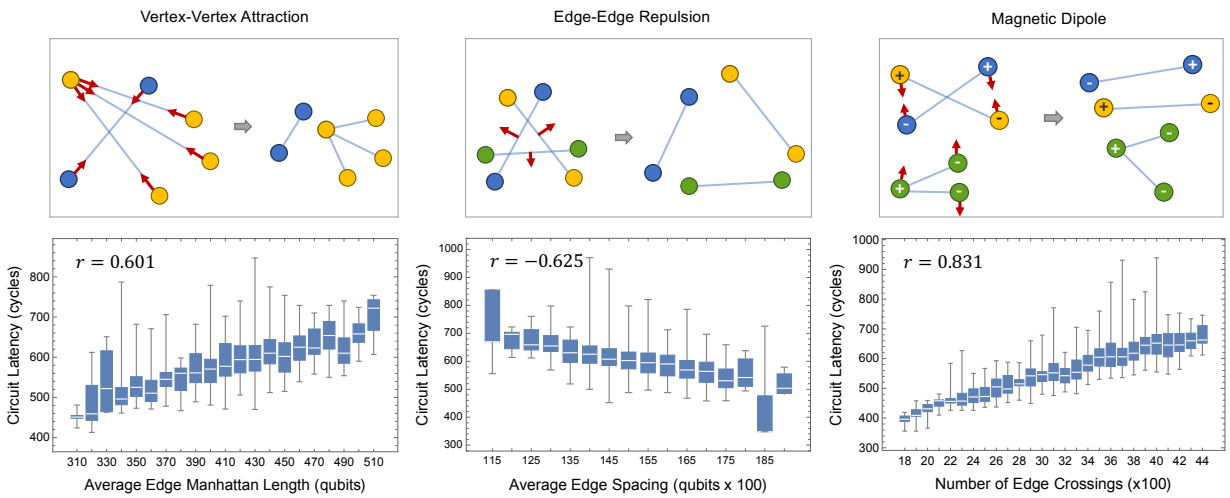


Figure 4.6: Heuristics (top) and metrics (bottom) used in our mapping algorithms, as described in Section 4.6.2 and 4.6.1 respectively. From left to right, edge length is minimized by vertex-vertex attraction, edge spacing is minimized by repulsion forces on the midpoints of edges, and edge crossings are minimized by applying rotational forces to edges emulating a magnetic dipole moment. For each metric, the correlation coefficient (r-value) is calculated across a series of randomized mappings of a distillation circuit, and latency is obtained through simulation, shown in bottom figures. The r-values of metrics with latency are $r = 0.601$, -0.625, and 0.831, respectively. The underlying intuition is that shorter edge length, larger edge spacing and fewer edge crossings will result in fewer braid conflicts and shorter overall latency.

4.6 Mapping

This section describes the impacts of qubit mapping decisions on the overall circuit overhead. Given a schedule we can define a *program interaction graph* as a graph $G = (V, E)$ where V is a set of logical qubits present in the computation, and E is a set of two-qubit interaction gates contained in the program (e.g. CNOT gates). By analyzing this graph, we can perform an optimized *mapping*, which assigns a physical location for each logical qubit $q \in V$.

Fig. 4.4a and Fig. 4.4b depict a single level and a two level factory, respectively, and distinct graph properties are available to analyze for each. The corresponding program that generates Fig. 4.4a is shown in Fig. 4.5. The single level factory is a planar graph. While the two level factory is constructed using many instances of the same single level factory, the requirement for states to be permuted between levels breaks the planarity of the resulting interaction graph. This has significant consequences, and we will leverage them in Section

4.7.

In order to execute state distillation most efficiently, we must minimize both the area required for the factory as well as the latency required to execute the circuit. Braid operations, while latency insensitive, still cannot overlap with one another. If an overlap is unavoidable, then one operation must stall while the other completes. As a consequence, we aim to minimize the number of these braid "congestions".

4.6.1 Heuristics for Congestion Reduction

Three common heuristics which we analyze for minimizing network congestion are: edge distance minimization, edge density uniformity, and edge crossing minimization. We see that they each correlate in varying degrees with actual circuit latency overhead for these quantum circuits, as shown in Fig. 4.6.

Edge Distance Minimization

The edge distance of the mapping can be defined as the Euclidean distance between the physical locations of each endpoint of each edge in the interaction graph. Intuitively, in classical systems network latency correlates strongly with these distances, because longer edges require longer duration to execute. As discussed in Section 5.2, for surface code braiding operations, there is no direct correspondence between single edge distance and single edge execution latency. However, longer surface code braids are more likely to overlap than shorter braids simply because they occupy larger area on the network, so minimizing the average braid length may reduce the induced network congestion.

Edge Density Uniformity

When two edges are very close to each other, they are more likely to intersect and cause congestion. Ideally, we would like to maximize the spacing between the edges and distribute

them on the network as spread-out and uniformly as possible. This edge-edge repulsion heuristic therefore aims to maximize the spacing between braid operations across the machine.

Edge Crossings Minimization

We define an edge crossing in a mapping as two pairs of endpoints that intersect in their geodesic paths, once their endpoint qubits have been mapped. These crossings can indicate network congestion, as the simultaneous execution of two crossing braids could attempt to utilize the same resources on the network. While the edge crossing metric is tightly correlated with routing congestion, minimizing it has been shown to be NP-hard and computationally expensive [83]. An edge crossing in a mapping also does not exactly correspond to induced network congestion, as more sophisticated routing algorithms can in some instances still perform these braids in parallel [47]. Some algorithms exist to produce crossing-free mappings of planar interaction graphs, though these typically pay a high area cost to do so [189].

Fig. 4.6 summarizes the correlation of each of these three metrics to surface code circuit latency.

4.6.2 Mapping Algorithms

With these metrics in mind, we explore two procedures designed to optimize mappings. First, we employ a local, force-directed annealing optimization technique designed to transform the optimized mappings of Fowler et al. [72] discussed in Section 7.2, specifically targeting optimization of the aforementioned metrics. Next, we compare this to a mapping procedure based upon recursive graph partitioning and grid bisection embedding.

Force-Directed Annealing

The full force-directed (FD) procedure consists of iteratively calculating cumulative forces and moving vertices according to these forces. Vertex-vertex attraction, edge-edge repulsion,

and magnetic dipole edge rotation are used to calculate a set of forces incident upon each vertex of the graph. Once this is complete, the annealing procedure begins to move vertices through the mapping along a pathway directed by the net force calculation. A cost metric determines whether or not to complete a vertex move, as a function of the combination of average edge length, average edge spacing, and number of edge crossings. The algorithm iteratively calculates and transforms an input mapping according to these force calculations, until convergence in a local minima occurs. At this point, the algorithm alternates between higher level *community* structure optimizations that either repulse all nodes within distinct communities away from one another, or attract all nodes within a single community together, which breaks the mapping out of the local minimum that it has converged to. This procedure is repeated until reaching a pre-specified maximum number of iterations.

Within an interaction graph, subsets of qubits may interact more closely than others. These groups of qubits can be detected by performing *community detection analysis* on an interaction graph, including random walks, edge betweenness, spectral analysis of graph matrices, and others [57, 86, 67, 110, 27, 60]. By detecting these structures, we can find embeddings that preserve locality for qubits that are members of the same community, thereby reducing the average edge distance of the mapping and localizing the congestion caused by subsets of the qubits.

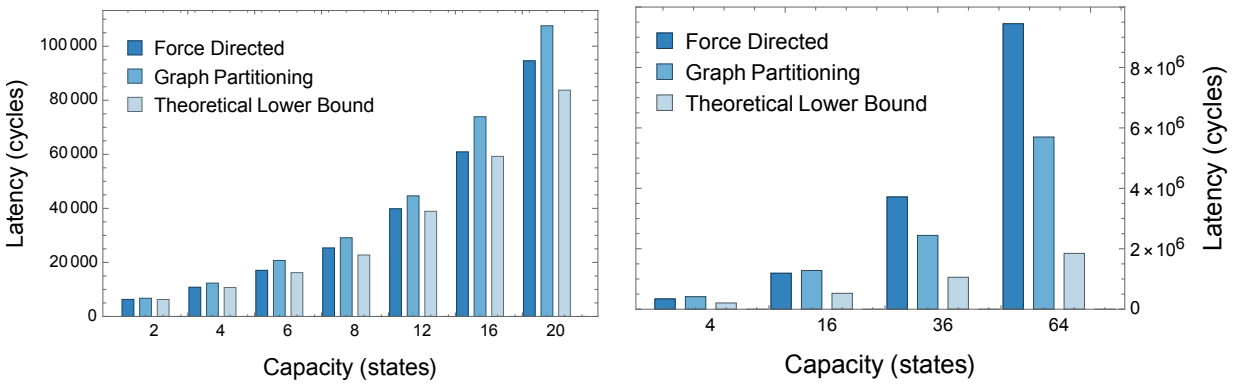
Edge Distance: To minimize the overall edge distance of the mapping, the procedure calculates the *centroid* of each vertex by calculating the effective "center of mass" of the neighborhood subgraph induced by this vertex, i.e. the subgraph containing only the vertices that are connected to this vertex, along with the corresponding edges. The center location of this set is calculated by averaging the locations of all of the neighbors, and this is assigned as the centroid for this vertex. This creates an attractive force on this vertex that is proportional in magnitude to the distance between the vertex and the centroid, as shown in the top-left panel in Fig. 4.6. Forces of this type are standard in graph drawing techniques [108].

Edge Density: In an attempt to optimize and uniformly distribute the edge density of

the mapping, repulsion forces are defined between each pair of distinct edges on the graph. Specifically, for each pair of edges, a repulsion force is created on the endpoints of magnitude inversely proportional to the square of the distance between the midpoints of the edges. This force law is reflected in many typical graph drawing techniques as well, that aim to uniformly distribute graph vertices and edges [142, 79].

Edge Crossings: Even though directly minimizing edge crossings in a graph is in general a difficult task to perform, we can approximate it by modeling each edge as a magnetic dipole moment, and the rotational forces applied on each edge will prefer (anti-)parallel orientations over intersecting ones, as shown in Fig. 4.6. North and south poles are assigned to every vertex in the graph, and attractive forces are created between opposing poles, while repulsive forces are added between identical poles. The assignment of the poles is done by producing a 2-coloring of the interaction graph. Notice that the graph is not always 2-colorable, and it usually is not. However, within each time step in the schedule, a vertex (qubit) can have degree at most 2, and is always acyclic. This is because we have a schedule that contains only 2-qubit gates and single-control multi-target CNOTs. Any two gates cannot be performed on the same qubit simultaneously, and the multi-target CNOTs will look like a vertex-disjoint path.

Community Structure Optimizations: To respect the proximity of the vertices in a detected community, we break up our procedure into two parts: firstly, impose a repulsion force between two communities such that they do not intersect and are well separated spatially; secondly, if one community has been broken up into individual components/clusters, we join the clusters by exerting attracting forces on the clusters. In particular, we use the KMeans clustering algorithm [120, 10] to pinpoint the centroid of each cluster within a community and use them determine the scale of attraction force for joining them.



(a) In single-level factories, both techniques can nearly optimally execute these circuits, even as capacity increases.

(b) In two-level factories, the difference between the theoretical lower bound and the attained circuit latencies widens.

Figure 4.7: Overall circuit latency obtained by graph partitioning embedding on single and two level distillation factories. Theoretical lower bounds are calculated by the critical path length of the circuits, and may not be physically achievable.

Recursive Graph Partitioning

To compare against the local force-directed annealing approach, we also analyzed the performance of a global grid embedding technique based upon graph partitioning (GP) [125, 20, 122]. In particular, we utilized a recursive bisectioning technique that contracts vertices according to a heavy edge matching on the interaction graph, and makes a minimum cut on the contracted graph. This is followed by an expanding procedure in which the cut is adjusted to account for small discrepancies in the original coarsening [121, 174]. Each bisection made in the interaction graph is matched by a bisection made on the grid into which logical qubits are being mapped. The recursive procedure ultimately assigns nodes to partitions in the grid that correspond to partitions in the original interaction graph.

The primary difference between these two techniques is that the former force-directed approach makes a series of local transformations to a mapping to optimize the metrics, while the graph partitioning approach can globally optimize the metrics directly.

Scalability Analysis

We can now compare the computational complexity of the two graph optimization procedures. Suppose we have an interaction graph of n vertices and m edges. Each iteration of the force-directed annealing procedure consists of three steps, vertex attraction, edge repulsion, and dipole moment rotation. In the worst case, the attraction forces are computed along each edge in $\mathcal{O}(m)$ time; the repulsion force computation requires $\mathcal{O}(m^2)$ time; rotations are calculated first by a DFS-style graph coloring and then by forces between vertices with $\mathcal{O}(n^2)$.

Graph partitioning requires recursively finding minimum weight cut, and partition the graph along the cut. Specifically, it requires $\log_2(n)$ recursive iterations, each of which is a min-cut algorithm on partitions of the graph that requires $\mathcal{O}(n+m)$ time, combining to $\mathcal{O}((n+m)\log_2(n))$ [121].

Performance Comparison

Fig. 4.7a and 4.7b indicate that, while both techniques perform well for single level factories, the global technique is much better at optimizing higher level factories. This is likely due to the local nature of the force-directed procedure, which is being used to transform the linear hand-optimized initial mapping of the factory. For higher level factories, this hand-optimized mapping incurs high overheads, and the local optimizations are only able to recover a portion of the performance proportional to the original mapping.

While the global graph partitioning technique works well in comparison with the local procedure, there is a widening performance gap between the resulting mapping and the critical resource volume, as factories grow in capacity and levels. This likely indicates that while the procedure is able to very effectively optimize small planar graphs, it has a more difficult time as the size and complexity of the graphs increase. In fact, single level factories have planar interaction graphs, and graph partitioning is able to optimize the mapping of these graphs nearly up to critical resource volume.

4.7 Hierarchical Stitching Method

We here present the outline of the iterative, synthesized optimization procedure that make use of the scheduling and mapping techniques we established earlier. To take advantage of the facts that most global optimization techniques (such as graph partitioning and force-directed annealing) work well on small planar graphs and that the circuit modules within each round form disjoint planar subgraphs, we develop a stitching scheme, as depicted in Fig. 4.8, that respects the hierarchical structure and internal symmetry of the multilevel block protocol while simultaneously optimizing for the previously discussed congestion heuristics.

As shown in Fig. 4.3, we perform optimizations iteratively on the interaction graph. In each iteration, our procedure is decomposed into two phases: (1) *inter-round* optimization that embeds and concatenates each module in the current round, and (2) *intra-round* optimization that stitches permutation edges and arranges modules in the next round.

4.7.1 Intra-Round Graph Concatenation

Starting with the first round of a multilevel factory, we use single-level optimization techniques (such as force-directed annealing or graph partitioning) to nearly optimally embed the *individual* planar modules. They are then *concatenated* together to form a full mapping of the first round of the factory circuitry. The concatenation scheme works well due to the fact that modules in a round do not interact with each other under block code protocol. Notice that putting barrier between rounds enables us to isolate and individually optimize for each round, as discussed in Section 4.5. Because the modules in each round of the factory are identical in schedule to those in all other rounds, the optimized graph partitioning embedding does not need to change for each round.

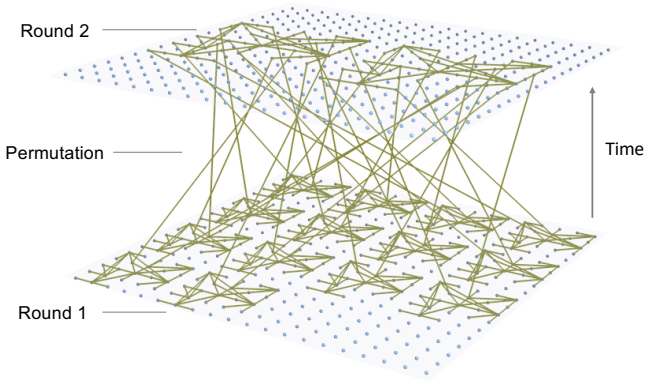
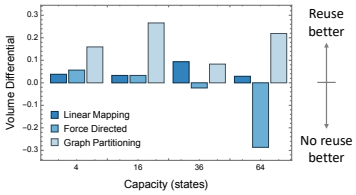


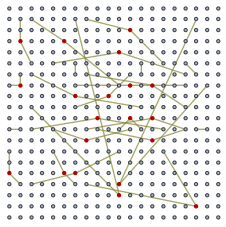
Figure 4.8: Embedding for a capacity $K = 4$, level $L = 2$ factory. The stitching procedure optimizes for each round to execute at nearly critical path length in latency, and optimizes for inter-round permutation step with force-directed optimizations.

Procedure	K	Linear	FD	GP
Latency	4	1.066	1.046	0.931
	16	1.203	1.203	0.913
	36	1.132	1.277	1.145
	64	1.216	1.611	0.978
Area	4	0.203	0.903	0.903
	16	0.804	0.804	0.804
	36	0.801	0.801	0.801
	64	0.798	0.799	0.799
Volume	4	0.964	0.944	0.841
	16	0.967	0.967	0.734
	36	0.906	1.023	0.917
	64	0.971	1.287	0.781

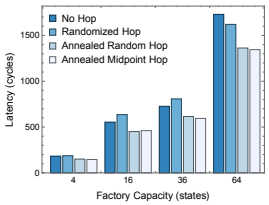
(a) Resource ratios comparing qubit reuse to non-reuse protocols



(b) Volume differentials between
qubit reuse and non-reuse protocols.



(c) Red dots show optimized intermediate destinations.



(d) Permutation latencies by no midpoint, Valiant, or annealed midpoints.

Figure 4.9: (a)-(b): Sensitivity of achievable quantum volumes by different optimization procedures. Shown is the percentage difference of the protocol with reusing (R) or without reusing (NR) qubits: (NR − R)/NR. Notably, reuse policy is a better for both the linear mapping and graph partitioning techniques, while no-reuse offers more flexibility for force-directed procedure to optimize. (c)-(d): Circuit latency specifically for the inter-round permutation step. Latency is reduced by 1.3x with Valiant-style intermediate destinations for each interaction, and using force-directed annealing to optimize their locations.

4.7.2 Inter-Round Permutation Optimization

The recursive block code structure requires that the output from lower levels of the factory be permuted and sent to new locations for the subsequent rounds. This can create highly-congested "permutation steps" in the circuit, where even though each round is scheduled and mapped nearly optimally, the cost to permute the outputs of one round to the inputs of the next round are quite high, as illustrated in the comparison of Fig. 4.9d with Fig. 4.10c. We therefore present the following sequence of procedures that target the inter-round communi-

cation/permutation overhead.

Qubit Reuse and Module Arrangement

The permutation edges in between two rounds are due to communications between the *output* qubits from the previous round and the *input* qubits in the next round. Given an optimal layout of the modules/blocks from the previous round, we know where the output states are located. Since all qubits except for the outputs are measured, error-checked, and then reinitialized by the time the next round starts, we can choose which regions of qubits to be reused for the next round, as long as for each module the following constraints are satisfied: (1) do not overlay a module on top of output qubits that are not supposed to be permuted to this particular module (see details about port assignment in 4.7.2), and (2) allocate enough qubits required by the code distance as discussed in 5.2.5. Fig. 4.9a and 4.9b show that reusing qubits benefits the linear and graph partitioned mapping techniques, while force-directed annealing prefers the flexibility added by not reusing qubits.

Port Reassignment

To avoid having correlated error in the inputs to the next round, each module in the next round must gather input states from different modules in the previous round, as shown in 5.2.5. Suppose one module from the next round wants a magic state from a previous-round module, when there are multiple outputs produced in that module, it does not matter which one you choose. Therefore, it leaves the optimization procedure to decide which output port to use, so as to minimize congestions in the permutation step.

Intermediate Hop Routing

Lastly, we employ a variation of the force-directed annealing algorithm from Section 4.6. Specifically, we introduce *intermediate destinations* between each output state from a prior

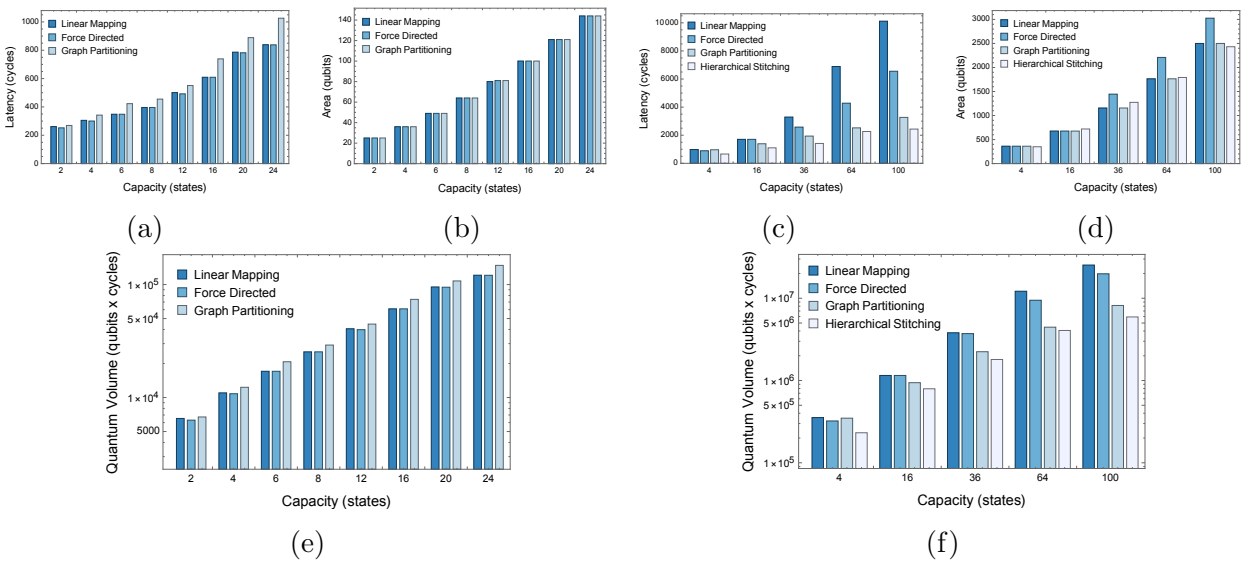


Figure 4.10: One and two level factory resource requirements. For the single level factory, we present latencies (4.10a), areas (4.10b), and achieved space-time volumes (4.10e). For the two-level factory, the right-hand side shows latencies (4.10c), areas (4.10d), and volumes (4.10f). All three optimizations are effective for reducing the overhead of single level factories. For two level factories, each procedure trades off space and time separately, resulting in the lowest achievable volume by that procedure. Hierarchical stitching is able to reduce space-time volume by 5.64x.

round and the input state to the next round, as depicted in Fig. 4.9. While Valiant routing with randomized intermediate destinations does not increase performance very significantly, we are able to use force-directed annealing based upon edge distance centroids, edge repulsion, and edge rotations in order to move the intermediate destinations into preferable locations.

This synthesized procedure is able to leverage the scheduling techniques of barrier insertion, combined with nearly optimal planar graph embedding performed by recursive graph partitioning, and force-directed annealing to obtain a significant resource reduction over any other optimization procedures.

	Level 1					Level 2				
Procedure	$K = 2$	4	8	10	24	$K = 4$	16	36	64	100
Random	1.11×10^4	1.82×10^4	5.43×10^4	6.40×10^4	2.70×10^5	–	–	–	–	–
Line(NR)	6.53×10^3	1.10×10^4	2.53×10^4	2.94×10^4	1.29×10^5	3.68×10^5	1.19×10^6	4.19×10^6	1.25×10^7	3.34×10^7
Line(R)	6.53×10^3	1.10×10^4	2.53×10^4	2.94×10^4	1.29×10^5	3.55×10^5	1.15×10^6	3.80×10^6	1.22×10^7	2.53×10^7
FD	6.30×10^3	1.08×10^4	2.53×10^4	2.88×10^4	1.21×10^5	3.22×10^5	1.15×10^6	3.72×10^6	9.45×10^6	1.98×10^7
GP	6.73×10^3	1.23×10^4	2.91×10^4	3.33×10^4	1.48×10^5	3.48×10^5	9.41×10^5	2.24×10^6	4.45×10^6	8.17×10^6
HS	–	–	–	–	–	2.32×10^5	7.93×10^5	1.80×10^6	4.06×10^6	5.93×10^6
Critical	6.28×10^3	1.07×10^4	2.27×10^4	3.03×10^4	1.12×10^5	1.82×10^5	4.48×10^5	8.85×10^5	1.53×10^6	2.43×10^6

Table 4.1: Quantum volumes required by factory designs optimized by: randomization (Random), linear mapping (Line) with and without qubit reuse (R, NR), force-directed (FD), graph partitioning (GP), and hierarchical stitching (HS).

4.8 Results

4.8.1 Evaluation Methodology: Simulation Environment

To perform evaluation of our methods, we implemented each configuration of the full Bravyi-Haah distillation protocol in the Scaffold programming language, and compiled this to gate-level instructions (e.g. Fig. 4.5). These instructions are fed into a cycle-accurate network simulator [112] that accurately executes the scheduling and routing of braids on a 2-dimensional surface-code qubit mesh. We extended both of these tools to support a multi-target CNOT gate. The simulator first schedules braids in parallel where the interaction graph allows. If braids intersect on the machine, the simulator inserts a stall to allow one braid to complete before the other. To perform scheduling, the simulator treats any data hazard (i.e. the presence of the same qubit in consecutive instructions) as a true dependency. This eliminates gate-level optimizations from being automatically performed, but it simultaneously allows for the introduction of "barrier" type gates. These are implemented by inserting a single gate involving all qubits of the machine (specifically a multi-target CNOT operation controlled on an extra qubit set to the zero state, and targeting all other qubits of the machine). Gate-level optimizations involving the commutativity relations of specific operations are performed by hand, independent of scheduling.

4.8.2 Single-Level Factory Evaluation

We notice first that the linear mapping procedure [72] performs well, even as the capacity of the factory increases. In Fig. 4.7a, we see that the linear mapping technique actually is able to approach the theoretical minimum required latency for each of these circuits. These mappings were specifically designed to optimize for these single level factories, which justifes these scaling properties.

Our proposed force-directed mapping approach described in section 4.6.2 is able to improve slightly from the linear mapping technique in most cases. This is due to the strong correlation of the metrics that the approach optimizes, to the realized circuit latency.

Graph partitioning techniques described in 4.6.2 underperform the linear mapping and force directed procedures, although they are still competent with respect to the theoretical minimum resource requirements. Because of the simplicity of the circuit and the targeted optimizations that were perform specifically for these small circuits, the advantage from the global nature of the graph partitioning method is significantly diminished.

The realistic circuit latency as executed in simulation, required circuit area, and corresponding quantum volume for single level magic state distillation factories are shown in Fig. 4.10a, 4.10b, and 4.10e. The best performing approach for each of the single level factories closely approximates the theoretical minimum latency and space-time volume required by these circuits. This is leveraged by our iterative procedure and used to ultimately achieve the most efficient circuit expression.

4.8.3 Multi-Level Factory Evaluation

Effects of Qubit Reuse Protocols

As anticipated, by electing to reuse qubits for later rounds in the distillation circuits, the overall circuit consumes less area at the cost of higher latency. Qubit reuse, for both the linear mapping and graph partitioning optimization methods, results in lower space-time

volume.

The force directed procedure actually achieves a lower volume when qubits are not reused. This is due to two factors introduced by qubit reuse. First, the average degree of the interaction graph has increased due to the introduction of false dependencies. This restricts the optimization procedure from being able to minimize the heuristics cleanly, as each qubit is more tightly connected to others, reducing the degrees of freedom in the graph. Second, there is more area in the graph, which widens the search space available for the procedure. With more possible configurations, the algorithm is more likely to find more optimized mappings.

Optimization Procedure Comparison

Fig. 4.10 shows the minimized space-time volumes achieved by each optimization procedure. While the linear mapping and force-directed procedures were able to nearly optimally map single level factories, the performance deteriorates significantly when moving to multi-level factories. In these factories, Hierarchical Stitching is able to outperform the other optimization strategies, as it synthesizes the best performing components of each.

We also considered both qubit reuse and non-reuse policies. The optimal combinations vary slightly for each procedure: the linear mapping and graph partitioning strategies always perform best with qubit reuse, while the force directed procedure performs best with qubit reuse for capacity 4 and 16 two level factories, and without qubit reuse for capacity 36 and beyond. This is due to the additional degrees of freedom that avoiding qubit reuse injects, as discussed above. The final results plots show these configurations.

In moving to multi-level factory circuits, even though there is significant modularity and symmetry in the factory, the introduction of the output state permutation from one level to the input states of the next introduces severe latency overheads. Without taking this into consideration, the linear mapping procedures suffer from large latency expansions in attempting to execute multi-level circuits, with the effect compounding as the size (output capacity) of the factory increases. Fig. 4.10f shows that the force-directed approach is able to

improve to a maximum reduction of ∼1.27x from these linear mappings, but is constrained by how poorly these mappings originally perform.

The graph partitioning technique is able to simultaneously optimize for the entirety of the multi-level circuit, including the inter-round communication steps. With all of this information, the technique is able to minimize interaction graph edge crossings and edge lengths, which results in a more efficient expression of the circuits overall for larger two level circuits. Smaller two level circuits are still dominated by the intra-round execution overheads, which are able to be effectively minimized by linear mapping and force directed techniques. Once multi-level factories become large enough (occurring in Fig. 4.10f at capacity 16), the inter-round effects begin to dominate. This is the point when graph partitioning techniques are able to outperform other methods.

The proposed hierarchical stitching technique is able to leverage the strengths of the force directed and graph partitioning methods to more effectively reduce resource consumption by mapping each round to near optimality and utilizing the same force-directed technique combined with the introduction of intermediate destinations to mitigate the overheads incurred by inter-round communication. Within all explored multi-level factory circuits, these optimizations further reduced resource consumption. In the largest case, a capacity 100 two level factory shows a 5.64x reduction in overall consumed quantum volume when moving from the linear mapping approach without reusing qubits, to hierarchical stitching.

4.9 Future Work

There are a number of immediate extensions to this work:

- *System-Level Performance.* We are studying the effect of higher-level factory optimizations on application performance. This includes analysis of resource distribution, comparison of factory system layout topologies, as well as architectures with prepared state buffers. The interaction with the Hierarchical Stitching procedure is currently

being analyzed.

- *Stitching Generalization.* Our proposed hierarchical stitching procedure can be applied to other hierarchical circuits, and to arbitrary circuits coupled with procedures that detect hierarchical sub-circuits. For example, we may extract sets of (planar) sub-divisions from the interaction graph and map each sub-division onto the 2-D surface, and perform permutations (`swap` gates) that patches the set of mappings together.

- *Teleportation vs. Lattice Surgery vs. Braiding.* Along the lines of [112, 156], we plan to explore the impacts of changing the surface code interaction style. Our proposed optimizations may likely change the trade off thresholds presented in [112].

- *Loss Compensation.* Typically in distillation protocols, when magic states are marked as defective they would be discarded and cause module failure. Future work would include implementing protocols that *compensates* the loss of those defective magic states by having back-up maintenance modules that feed high-fidelity states to ensure the completion of the distillation round.

- *Area Expansion.* It is possible to expand the utilized area for these distillation circuits and reduce the execution latency. Our force directed procedures work well with additional area, so this may reduce overall consumed space-time volume.

4.10 Conclusion

Error correction is the largest performance bottleneck of long-term quantum computers, and magic-state distillation is the most expensive component. Known optimized scheduling and mapping techniques for state distillation circuits tend to work well for small, single level factories, but quickly incur large overheads for larger factories. We have proposed a technique that synthesizes mapping and scheduling optimizations to take advantage of the unique efficiencies of each, which allows for a significant 5.64x reduction in the realistic

space-time volume required to implement multi-level magic state distillation factories. Global optimizations like graph partitioning and force-directed annealing work well, but leveraging structure of the block code circuitry combined with the specific strengths of both graph partitioning and force-directed annealing allows for the most improvement, resulting in large factors of resource reduction overall.

CHAPTER 5

RESOURCE OPTIMIZED QUANTUM ARCHITECTURES FOR SURFACE CODE IMPLEMENTATIONS OF MAGIC-STATE DISTILLATION

Quantum computers capable of solving classically intractable problems are under construction, and intermediate-scale devices are approaching completion. Current efforts to design large-scale devices require allocating immense resources to error correction, with the majority dedicated to the production of high-fidelity ancillary states known as magic-states. Leading techniques focus on dedicating a large, contiguous region of the processor as a single "magic-state distillation factory" responsible for meeting the magic-state demands of applications.

In this work we design and analyze a set of optimized factory architectural layouts that divide a single factory into spatially distributed factories located throughout the processor. We find that distributed factory architectures minimize the space-time volume overhead imposed by distillation. Additionally, we find that the number of distributed components in each optimal configuration is sensitive to application characteristics and underlying physical device error rates. More specifically, we find that the rate at which T-gates are demanded by an application has a significant impact on the optimal distillation architecture. We develop an optimization procedure that discovers the optimal number of factory distillation rounds and number of output magic states per factory, as well as an overall system architecture that interacts with the factories. This yields between a 10x and 20x resource reduction compared to commonly accepted single factory designs. Performance is analyzed across representative application classes such as quantum simulation and quantum chemistry.

5.1 Introduction

Quantum computers promise to provide computational power required to solve classically intractable problems and have significant impacts in materials science, quantum chemistry,

cryptography, communication, and many other fields. Recently, much focus has been placed on constructing and optimizing Noisy Intermediate-Scale Quantum (NISQ) computers [178], however over the long term quantum error correction will be required to ensure that large quantum programs can execute with high success probability. Currently, the leading error correction protocol is known as the surface code [53, 69], which benefits from low overheads in terms of both fabrication complexity and amount of classical processing required to perform decoding.

A common execution model of machines protected by surface code error correction requires a process called *magic-state distillation*. In order to perform universal computation on a surface code error corrected machine, special resources called *magic states* must be prepared and interacted with qubits on the device. This process is very space and time intensive, and while much work has been performed optimizing the resource preparation circuits and protocols to make the distillation process run more efficiently internally [34, 93, 116, 73, 54], relatively little focus has been placed upon the design of an architecture that *generates* and *distributes* these resources to a full system.

This study develops a realistic estimate of resource overheads of, and examines the trade-offs present in, the architecture of a system that prepares and distributes magic states. In particular, instead of using a single large factory to produce all of the magic states required for an application, the key idea of our work is to distribute this demand across several smaller factories that together produce the desired quantity. We specifically characterize these types of distributed factory systems by three parameters: the total number of magic states that can be produced per cycle, the number of smaller factories on the machine, and the number of distillation rounds that are executed by each factory.

The primary trade-off we observe is between the number of qubits (area/space) and the amount of time (latency) spent in the system: we can design architectures that use minimal area but impose large latency overheads due to lower magic-state output rate, or we can occupy larger amounts of area dedicated to resource production aiming to maximally alleviate

application latency. The two metrics, space and time, are equally important as it is easy to build small devices with more gates or large devices with few gates. This concept is closely related to the idea of "Quantum Volume" [25], when machine noise and topologies are taken into consideration. To capture the equal importance of both of these metrics, we use a space-time product cost model in which the two metrics simply multiply together. This model has been used elsewhere in similar analysis [73, 54, 166, 112].

Figure 5.1 illustrates the opposing trends for space and time when we increase the magic-state production rate. Our goal is to find the "sweet spot" on the combined space-time curve, where the overall resource overhead is at its lowest.

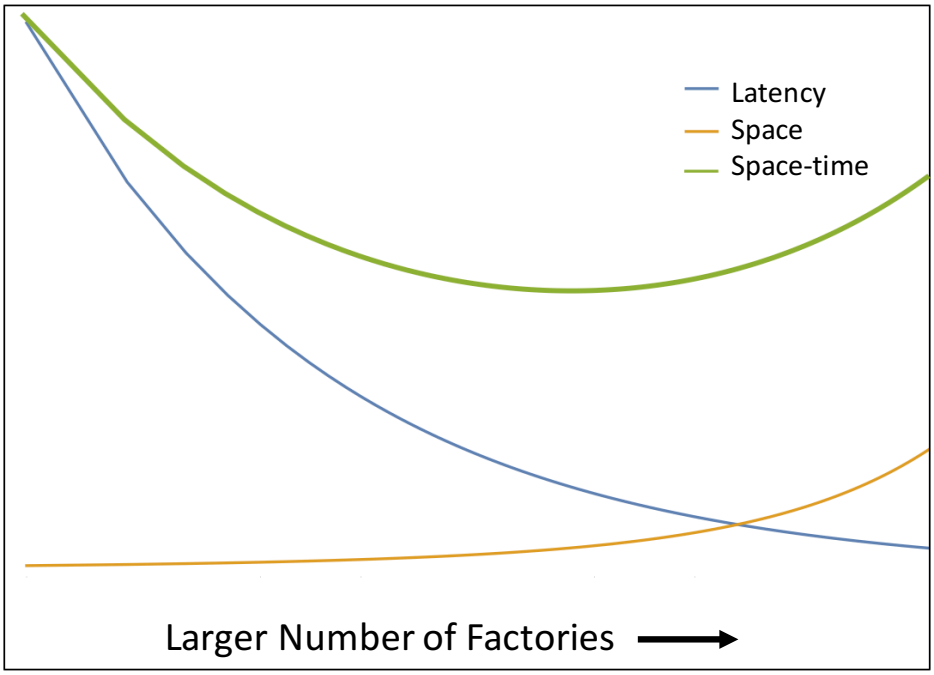


Figure 5.1: Space and time tradeoffs exist for distributions of resource generation factories within quantum computers. These trends are shown assuming same total factory output capacity. By explicit overhead analysis, we can discover optimal space-time volume design points.

In summary, this paper makes the following contributions:

1. We present precise resource estimates for implementing different algorithms with magic-state distillation on a surface code error corrected machine. We derive the estimates

from modeling and simulating the generation and distribution of magic states to their target qubits in the computation.

2. We quantify the space and time trade-offs of a number of architectural configurations for magic-state production, based on design parameters including the total number of factories, total number of output states these factories can produce, and the desired fidelity of the output magic states.

3. We study different architectural designs of magic-state distillation factory, and present an algorithm that finds the configuration that minimizes the space-time volume overhead.

4. We highlight the nontrivial interactions of factory failure rates and achievable output state fidelity, and how they affect our design decisions. We analyze the sensitivity of these optimized system configurations to fluctuations in underlying input parameters.

5. We discover that dividing a single factory into multiple smaller distributed factories can not only reduce overall space-time volume overhead but also build more resilience into the system against factory failures and output infidelity.

The rest of the paper is structured as follows. In Section 5.2, a basic background of quantum computation, error correction, magic-state distillation and the Bravyi-Haah distillation protocol, as well as the block-code state-distillation construction are described. Section 7.2 describes previous work in this area. Sections 5.4 and 5.5 discuss important space and time characteristics of the distillation procedures that we consider, and derive and highlight scaling behaviors that impact full system overhead analysis. Section 5.6 describes in detail how these characteristics interact, and shows how these interactions create a design space with locally optimal design points. Section 5.7 details the system configurations we model, describes a novel procedure for discovering the optimal design points, and discusses the simulation techniques used to validate our model derivations. Section 7.7 shows our results and the explains the impacts of optimizing these designs. Sections IV and 5.10 conclude and discuss ideas to be pursued as future work.

5.2 Background

5.2.1 Quantum Computation

The idea of quantum computation is to use quantum mechanics to manipulate information stored in two-level physical systems called quantum bits (qubits). In contrast to a bit in a classical machine, each qubit can occupy two logical states, denoted as $|0\rangle$ and $|1\rangle$, as well as a linear combination (superposition) of them, which can be written as $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, where α, β are complex coefficients satisfying $|\alpha|^2 + |\beta|^2 = 1$.

It is sometimes useful to visualize the state of a single qubit as a vector on the bloch sphere [26, 154], as we can rewrite the state $|\psi\rangle$ in its spherical coordinates as $|\psi\rangle = \cos(\theta/2)\,|0\rangle + \exp(i\phi)\sin(\theta/2)\,|1\rangle$. Any operations (called quantum logic gates) performed on single qubit can thus be regarded as rotations by an angle φ along some axis $\hat{n}$, denoted as $R_{\hat{n}}(\varphi)$. In this paper we will focus on some quantum gates that are commonly used in algorithms, such as the Pauli-X gate ($X \equiv R_x(\pi)$), Pauli-Z gate ($Z \equiv R_z(\pi)$), Hadamard gate ($H \equiv R_x(\pi)R_y(\pi/2)$), S gate ($S \equiv R_z(\pi/2)$), and T gate ($T \equiv R_z(\pi/4)$). For multi-qubit operations, we will consider the most common two-qubit gate called controlled-NOT (CNOT). It has been shown [16] that the above mentioned operations form a *universal* gate set, which implies that any quantum operations can be decomposed as a sequence of the above gates.

As quantum logic gates require extremely precise control over the states of the qubits during execution, a slight perturbation of the quantum state or a minor imprecision in the quantum operation could potentially result in performance loss and, in many cases, failure to obtain the correct outcomes. In order to maintain the advantage that quantum computation offers while balancing the fragility of quantum states, quantum error correction codes (QECC) are utilized to procedurally encode and protect quantum states undergoing a computation. One of the most prominent quantum error correcting codes today is the surface code [53, 69].

5.2.2 Surface Code

In a typical surface code implementation, physical qubits form a set of two-dimensional rectangular arrays (of logical qubits), each of which performs a series of operations only with its nearest neighbors. A logical qubit, under this construction, is comprised of a tile of physical qubits, and these tiles interact with each other differently according to different logical operations. These interactions on the grid create the potential for communication-imposed latency, as routing and logical qubit motion on the lattice must be accomplished.

An important parameter of the surface code is the *code distance* d. Larger code distance means a larger tile for each logical qubit. The precise number of physical qubits required in each tile also depends on the underlying surface code implementation. Most common implementations assume a logical qubit of distance d requires $\sim d^2$ physical qubits [69, 106]. Code distance also determines how well we can protect a logial qubit. The logical error rate P_L of a logical qubit decays exponentially in d. More precisely:

$$P_L \sim d(100\epsilon_{in})^{\frac{d+1}{2}} \tag{5.1}$$

where ϵ_{in} is the underlying physical error rate of a system [73].

In particular, this work will focus on two relatively expensive operations on surface code, namely the logical CNOT gate and the logical T gate. Our overhead analysis will hold regardless of the underlying technology, e.g. superconducting or ion-trap implementations. Earlier work [112] has also performed such analysis with technology-independent frameworks. Firstly, a logical CNOT between two qubits can be expensive, because the two logical qubits can be located far apart on the lattice and long-distance interaction is achieved by the *topological defect braiding* methodology. Secondly, a logical T gate can also be costly because it requires some ancillary state to be procedurally prepared in advance, called the *magic-state distillation*.

CNOT Braiding

A *braid* is a path in the surface code lattice, or an area where the error correction mechanisms have been temporarily disabled and where no other operations are allowed to use. In other words, braids are not allowed to cross. A logical qubit can be entangled with another if the braid pathway encloses both qubits, where enclosing means extending a pathway from source qubit to target qubit and then contracting back via a (possibly different) pathway. It is important to note that these paths can extend up to arbitrary length in constant time, simply by disabling all area covered by the path in the same cycle. Furthermore, each path must remain open for a constant number of surface code cycles to establish fault tolerance. More precisely, one CNOT braid takes $T_{cnot} = 2d + 2$ cycles to be performed fault tolerantly [69, 112].

T Magic-States

Now T (and S) gates, as described earlier, are necessary for universal quantum computation, and yet are very costly to implement on the surface code. For simplicity of analysis, we assume all S gates will be decomposed into two T gates, because of their rotation angle relationship. This is potentially an overestimate of the actual gate requirements, as it is also possible to perform an S gate via separate distillation of a different type of magic state. We are also aware of another surface code implementation that allows for S gate to be executed without distillation [144]. These techniques have different architectural implications which are outside the scope of the analysis of this work.

To execute these gates, an ancillary logical qubit must be first prepared into a special state, known as the *magic state* [35]. Once prepared, this magic-state is to be interacted with the target qubit as in [69], via a probabilistic circuit involving the magic state and between 1 or 3 CNOT braids, each with probability 1/2. The extra 2 CNOTs are required to perform a corrective S gate in the case that the probabilistic circuit fails, which we assume to be consisting of 2 CNOT braids. This circuit is called the state injection circuit. We can

therefore write the expected latency of a T gate as

$$\mathbb{E}[T_t] = T_{cnot} + \frac{1}{2}(2 * T_{cnot}) = 4d + 4 \tag{5.2}$$

where we use T_t to denote latency of a T gate and T_{CNOT} as latency of a CNOT gate.

Since the task of preparing these states is a repetitive process, it has been proposed that an efficient design would dedicate specialized regions of the architecture to their preparation [200, 118]. These *magic-state factories* are responsible for creating a steady supply of low-error magic states. The error in each produced state is minimized through a process called *distillation* [34], which we will introduce in detail in section 5.2.4.

5.2.3 T-Gates in Quantum Algorithms

Among the different classes of quantum algorithms, quantum simulation and quantum chemistry applications have drawn significant attention in recent years due to the promises they show in transforming our understanding of new and complex materials, while still potentially remaining tractable in near-term intermediate-size machines [152, 13, 130, 224, 119].

The benchmark algorithms studied in this work include the *Ground State Estimation* (GSE) [224] of the Fe_2S_2 molecule and the *Ising Model* (IM) [19] algorithms. They are representative applications for the purpose of this study as they present very different demand characteristics for T gate magic states. A more detailed description of T gate distributions in these two algorithms can be found in section 5.5.1. Here we list in Table 5.1 the two benchmarks alongside with some of their T gates statistics, namely the number of qubits (n_{qubits}), total T count (T_{count}), total schedule length (L), average T gates per time step (T_{avg}), standard deviation of T gates per time step (T_{std}), and maximum T gates per time step (T_{peak}).

The Ising Model and Ground State Estimation applications, and others in the same application class, have a predictable structure. Contemporary methods to simulate quantum

Application	n_{qubits}	T_{count}	L	T_{avg}	T_{std}	T_{peak}
IM	500	9068348	20589	440	107	778
GSE	5	775522	546708	1.419	1.464	12

Table 5.1: T gate statistics in the Ising Model (IM) and Ground State Estimation (GSE) benchmarks. For our analysis, we consider a 500-qubit spin chain in our IM simulation, and we simulate a small molecule in GSE comprised of 5 spin orbital states. The reason T_{peak} for IM can be more than the number of qubits is because in this calculation every S gate in the application has been decomposed into 2 T gates.

mechanical systems employ Trotter decomposition [210] to digitize the simulation, which involves large numbers of structurally identical Jordan-Wigner Transformation circuits [22], each of which involves a series of CNOT gates (called the "CNOT staircase") followed by a controlled rotation operation. This arbitrary-angle rotation will often be decomposed to sequences of H, S, and T operations in a procedure called gate synthesis [186].

Take as an example finding molecular ground state energies of the molecule Fe_2S_2 requires approximately 10^4 Trotter steps for "sufficient" accuracy, each comprised of 7.4×10^6 rotations [223]. Each of these controlled rotations can be decomposed to sufficient accuracy using approximately 50 T gates per rotation [131]. All of this can amount to a total number of T gates of order 10^{12}, which is also the number of prepared magic-states needed. In these types of applications, magic-state distillation will be responsible for between $50\% - 99\%$ of the resource costs when executing an error-corrected computation [54]. Because of this, the number of T gates present in an algorithm is often used as a metric for assessing the quality of a solution [194, 8].

5.2.4 Bravyi-Haah Distillation Protocol

In order to execute T gates fault tolerantly, an interaction is required between a target logical qubit and an ancillary magic state qubit. The fidelity of the operation is then tied to the fidelity of the magic state qubit itself, which requires that magic states are able to be reliably produced at high fidelity. This is achieved through procedures known as distillation protocols.

Distillation protocols are circuits that accept as input a number of potentially faulty

raw magic states (n) and output a smaller number of higher fidelity magic states (k). The input-output ratio $n \to k$ is generally used to assess the efficiency of a protocol. Because many distillation protocols are extremely resource-intensive, a key design issue of quantum architectures is to optimize them.

In this work we restrict our focus to a popular low-overhead distillation protocol known as the Bravyi-Haah distillation protocol that has received much attention in the field recently [116, 73, 156]. Here we describe in detail the process for preparing and distilling the magic-states. Bravyi-Haah state distillation circuits [34] take as input $3k + 8$ low-fidelity states, and output k higher fidelity magic-states, and thus are denoted as the $3k + 8 \to k$ protocol. Notably, if the raw input (injected) states are characterized by error rate ϵ_{inject} (which could be different from the physical input error rate ϵ_{in} as in equation 5.1 depending on hardware implementations), the output state fidelity is improved with this procedure to:

$$\epsilon_{\text{output}} = (1 + 3k)\epsilon_{\text{inject}}^2, \tag{5.3}$$

or in other words, a second-order suppression of error.

This imposes a tolerance threshold on the underlying input error rate that can be precisely written as:

$$\epsilon_{\text{thresh}} \approx \frac{1}{3k + 1} \tag{5.4}$$

because when $\epsilon_{\text{inject}} \geq \epsilon_{\text{thresh}}$, the output error rate is no better than where we started before distillation.

Moreover, this process is imperfect. For any given implementation of this circuitry, the true yield could be lower than expected. The success probability of the protocol that attempts to output k high fidelity states is, to the highest order, given by:

$$P_{\text{success}} \approx 1 - (8 + 3k)\epsilon_{\text{inject}}. \tag{5.5}$$

In performing a rigorous full system overhead analysis, these effects will become extremely significant.

5.2.5 Block Codes

In certain types of applications, the second-order error suppression achieved by single round of Bravyi-Haah distillation is not enough. To overcome this, multiple rounds (also referred to as *levels* in our work) of the distillation protocol can be concatenated to obtain higher and higher output state fidelity.

To ensure successful execution of a program, systems must be able to perform all of the gates in the computation with an expected value of logical gate error rate less than 1. So the success probability desired for a specific application (P_s) relates to the required logical error rate per gate P_L as follows:

$$P_L \leq \frac{P_s}{N_{\text{gates}}} \tag{5.6}$$

where N_{gates} is the number of logical gates in the computation. P_L therefore sets a bound on the fidelity of generated magic states. Many circuits contain of order 10^{10} logical gates or more [223], while physical error rates may scale as poorly as 10^{-3} [73]. In these cases, clearly squaring the input error rate will not achieve the required logical error rate to execute the program. Instead, we can *recursively* apply the Bravyi-Haah circuit ℓ times, with permutations of the intermediate output states in between distillation rounds. Throughout this work, we use the terminology "round" and "level" to both refer to a single iteration of the Bravyi-Haah distillation protocol within a factory. Constructing high fidelity states in this fashion is known as Block Code State Distillation [116]. As shown in Figure 5.2, realizing Bravyi-Haah block code protocols would require $6k + 14$ total logical qubits [156].

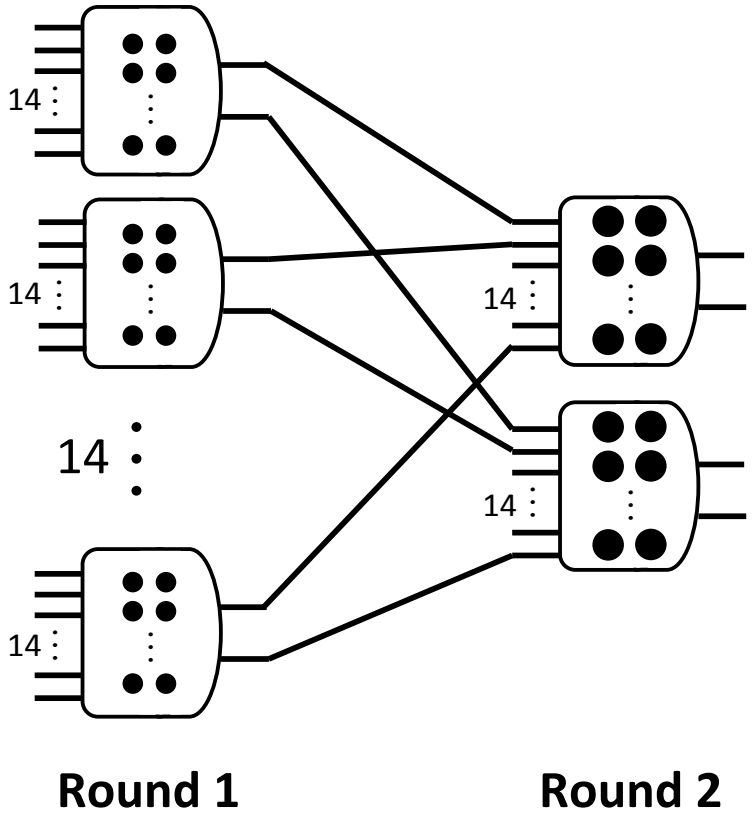


Figure 5.2: The recursive structure of the block code protocol. Each block represents a module for Bravyi-Haah $(3k+8) \to k$ protocol, and lines indicate the magic-state qubits being distilled, and dots indicates the extra $3k+6$ ancillary qubits used, totaling to $6k+14$. This figure shows an example of 2-level block code with $k = 2$. So this protocol takes in total $(3k+8)^2 = 14^2$ states, and outputs $k^2 = 4$ states with higher fidelity. The qubits (dots) in round 2 are drawn at bigger size, indicating the larger code distance d required to encode the logical qubits, as they have lower error rate than in the previous round [156].

Magic-State Factory Error and Yield Scaling

To perform a rigorous full system overhead analysis, it is necessary to quantify the behavior of multi-level block code factories in terms of output state fidelity and production rate. By construction, the error rate of the produced magic-states will be squared after each round. So the final output states error rate after ℓ rounds of distillation will be $\sim \epsilon_{\text{inject}}^{2^\ell}$.

Since the output states from the previous round will be fed into the next round, the success probability of a distillation module at round r depends on the output error rate of the previous round ϵ_{r-1}, i.e. $P_{\text{success}}^{(r)} = 1 - (3k+8)\epsilon_{r-1}$. The success probability for the

entire ℓ-level factory will be explicitly derived later in Section 5.4.

Magic-State Factory Area Scaling

Within any particular round r of an ℓ-round magic-state factory (where $1 \leq r \leq \ell$), the required number of *physical* qubits defines the space occupied by the factory during that round. However, we will often use *logical* qubit as unit area, since translating to physical qubits will simply pick up a d_r^2 multiplicative factor as shown in section 5.2.2.

In general, any particular round requires several modules each comprised of several distillation protocol circuits. A generic $n \to k$ protocol, under a ℓ-level block code construction, will need a total number of protocols as follows:

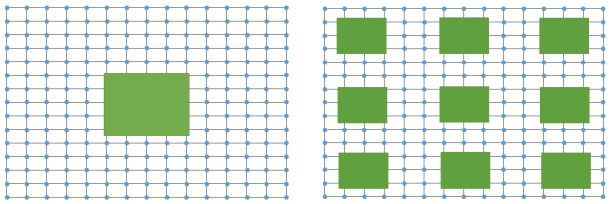
$$N_{\text{distill}} = \sum_{r=1}^{\ell} N_r = \sum_{r=1}^{\ell} k^{r-1} n^{\ell-r} \tag{5.7}$$

Magic-State Factory Time Overhead

Each round of distillation can be shown to require $11d_r$ number of surface code cycles[156]. Suppose d_r is the code distance for round r (which depends upon the input and output error rates), we arrive at the total time to execute full distillation as:

$$T_{\text{distill}} = 11 \sum_{r=1}^{\ell} d_r \tag{5.8}$$

A full assessment of the area and time costs under our proposed architecture designs,will be presented in more detail in Section 5.4 and Section 5.5. Specifically, we discuss how factory capacity, distillation rounds of each factory, and the input physical error rate all affect the output state yield rate and resulting space and time overhead.



(a) Single unified factory with *large* capacity

(b) A number of distributed factories, each with *smaller* capacity

Figure 5.3: The concept of a unified versus distributed factory architecture, embedding factories (green blocks) within computational surface code region (blue circles).

5.3 Related Work

A number of prior work has been focused upon designing efficient magic-state distillation protocols [34, 9, 150, 35]. There are also some work that aim to concatenate different protocols together to reduce the overall cost or improve output rate and fidelity [116, 41]. The problem of scheduling and mapping the distillation circuit is tackled in this work [54] by taking advantages of the internal structures in the distillation protocol, and by minimizing CNOT-braid routing congestions. The aim of that work is also to more effciently implement the distillation process, which is different from ours, as we instead aim to optimize a full system architecture built around these protocols and construct factory arrangements that efficiently deliver output magic states to their intended target qubits.

Prior work on this subject has often assumed either that magic states will be prepared offline in advance [73, 118], or that the production rate is set to keep up with the *peak* consumption rate in any given quantum application, and any excess states will be placed in a buffer [156, 211]. This paper operates with the different assumption that magic-state factories will be active during the computation, and states will not be able to be prepared offline or in advance. We do this to characterize the performance of the machine online, and introduce the complexity of resource state distribution throughout the machine, a problem that has been studied well in classical computing systems but has received less of a focus in this domain.

Parameter	Descriptions	Parameter	Descriptions
K	Factory total capacity	n	Number of input states in distillation protocol
X	Number of distributed factories	k	Number of output states in distillation protocol
ℓ	Block-code levels of a factory	N_{r}	Number of protocols at round r under block-code
r	Distillation round, $1 \leq r \leq \ell$	K_{output}	Number of effective output magic-states due to yield rate
d	Surface code distance	T_{distill}	Time to execute one full iteration of distillation
$P_{\mathrm{s}}, P_{\mathrm{success}}$	Target success probability	T_{t}	Time to deliver magic-state to target qubit
P_L	Logical fidelity	n_{distill}	Distillation iterations to support one timestep of a program
$\epsilon_{\mathrm{inject}}$	Physical error rate of raw magic-state	A_{factory}	Total area of factories (in physical qubits)
$\epsilon_{\mathrm{in/target/r}}$	Physical error rate at input, at output, or at round r		

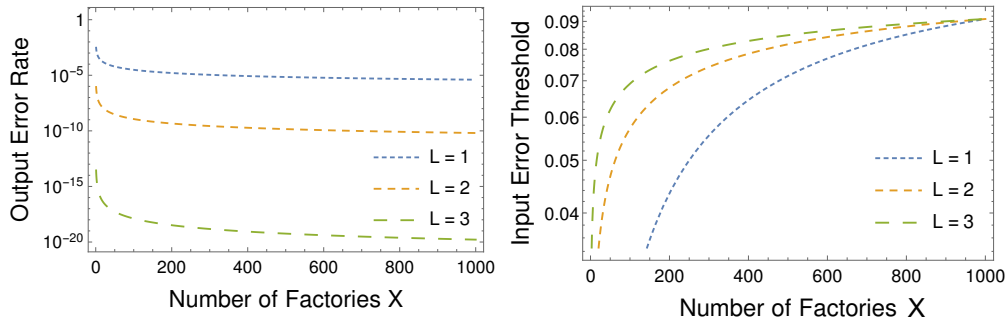
Table 5.2: List of system parameters involved in the analysis and the optimization procedure.

Other works closely related to architectural design optimized the ancilla production factories that operate in different error correcting codes [162, 111], or analyzed the overhead of CNOT operations which dominate other classes of applications like quantum cryptography and search optimization [112]. Our work focuses instead on quantum chemistry and simulation applications that are likely to represent a large proportion of quantum workloads in both the near and far term.

5.4 Factory Area Overhead

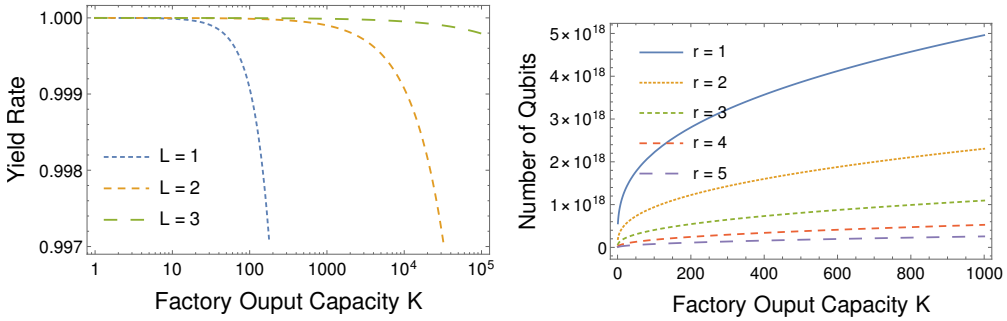
To describe a magic-state distillation factory, we first make a distinction between a factory *cycle* and a distillation *round* or *level*. A distillation round refers to one iteration of the distillation protocol, a subroutine that is repeated ℓ times for a particular factory. A cycle refers to the total time required for the factory to operate completely, taking n input states and creating k^{ℓ} output states. All ℓ distillation rounds are performed during a cycle.

A magic-state distillation factory architecture can be characterized by three parameters: the total number of magic states that can be produced per distillation cycle K, number of factories on the lattice X, and the total number of distillation rounds that are performed per cycle ℓ. For simplicity, we assume uniform designs where all K output states are to be divided equally into X factories, all of which operate with ℓ rounds of distillation. We now analyze the relationships presented in Section 5.2 to derive full factory scaling behaviors with respect to these architectural design variables. These behaviors interact non-trivially, and lead to space-time resource consumption functions that show optimal design points.



(a) Error rate attainable by number of factories

(b) Error rate tolerable by number of factories



(c) Yield rate of L-level factory with capacity K

(d) Area scaling within each round of a 5-level factory

Figure 5.4: (a) Higher fidelity output states are achievable with increasing number of factories at a fixed output capacity. (b) Increasing the number of factories in an architecture allows for higher tolerance of input physical error rates. (c) Increasing factory output capacity puts pressure on the factory yield rate, and increasing the number of levels pushes the yield dropoff point. (d) Maximum area to support multi-level factory is required of the lowest level of the factory, all higher levels require less area support.

5.4.1 Role of Fidelity and Yield in Area Overhead

First we examine the fidelity of the produced magic-states that is attainable with a given factory configuration, along with expected number of states that will in fact be made available. Once again, we use the terminology "round" and "level" to both refer to a single iteration of the Bravyi-Haah distillation protocol within a factory. Applying the block code error scaling relationship described by equation 5.3 recursively, as the number of total rounds (ℓ) of a magic-state factory increases, the output error rates attainable scale double-exponentially with the total number of rounds in a factory: ℓ. In fact, for a given round r (between 1 and ℓ) of a factory, the explicit form of the output error rate can be written by directly applying

r copies of equation 5.3:

$$\epsilon_r = (1 + 3(K/X)^{1/\ell})^{2^r - 1} \epsilon_{\text{inject}}^{2^r} \tag{5.9}$$

where (K/X) denotes the capacity of each factory on a lattice.

The yield rate of a particular factory can be expressed as a product of the yield rate functions describing each individual round, as in equation 5.5. The effective output capacity can be written as the product of the success probabilities of all ℓ rounds of a factory as:

$$K_{\text{output}} = K \cdot \prod_{r=1}^{\ell} \left[(1 - (3(K/X)^{1/\ell} + 8)\epsilon_{r-1} \right] \tag{5.10}$$

Here K_{output} refers to the realized number of produced states after adjusting for yield effects, while K refers to the desired or specified number of output states. Equation 5.10 actually imposes a *yield threshold* on the system. For a given K, X, and ℓ, a system will have a maximum error rate which, if exceeded, will cause the factory to malfunction and stop producing states reliably. This threshold can be seen by examining the product term, and noting that yield must be positive in order to produce any states. The terms in the sequence of equation 5.10 are decreasing in magnitude, so the threshold is determined by the leading term which requires: $1 - (3(K/X)^{1/\ell} + 8)\epsilon_{\text{inject}} > 0$, and thus:

$$\epsilon_{\text{thresh}} < \frac{1}{3(K/X)^{1/\ell} + 8} \tag{5.11}$$

Figure 5.4c shows the yield rate scaling behavior of single factories of consisting of $\ell = 1, 2, 3$ with fixed $X = 1$. In order to reliably produce some fixed amount of states, the yield effects determine the required number of rounds of distillation that must be performed. On the other side, any given number of distillation rounds has a maximum output capacity K for which the expected number of produced states becomes vanishingly small. Increasing the number of distillation rounds will increase the maximum supportable factory capacity.

5.4.2 Full Area Costs

We now use these relationships to derive the true area scaling of these factories. For all ℓ level factories, the area of the first round exceeds the area required for all other rounds. Using this as an upper bound, we can write the area required for a specific round explicitly in terms of physical qubits as:

$$A_r = X \cdot k^{r-1}(3k+8)^{\ell-r}(6k+14) \cdot d_r^2 \tag{5.12}$$

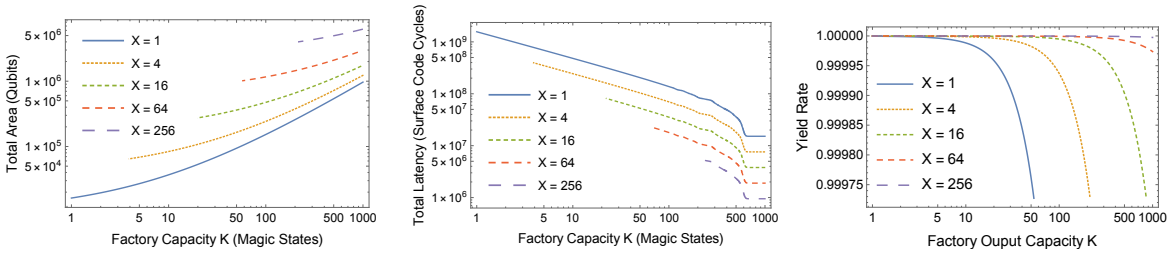
$$\leq X(3k+8)^{\ell-1}(6k+14) \cdot d_1^2 \tag{5.13}$$

Where $k \equiv (K/X)^{1/\ell}$. The inequality in the last line arises due to the fact that the first round always uses the largest area by block-code construction, i.e. $A_r \leq A_1$ for all $1 \leq r \leq \ell$. Here we have used several relationships, namely that the total number of protocols and modules scales as in equation 5.7, a single protocol requires $6k + 14$ logical qubits [156], and the area of a single logical surface code qubit scales as d^2 [106].

Although in an aggressively optimized factory design then, one could conceivably save space within the distillation procedure by utilizing the space difference between successive rounds of distillation for other computation, we will assume in this work that this cannot be done, and instead the first round area of any given factory defines the area required by that factory over the length of its entire operation, and *locks out* the region for distillation only. As a result, Figure 5.5a describes the scaling of factory area both by increasing output capacity and increasing the total number of factories.

5.5 Factory Latency Overhead

This section presents a systematic study of the time overhead of realizing magic-state distillation protocols. First, we will examine the characteristics of the T gate demand in our benchmark programs, by introducing the concept of the T distribution. Next, we will study the latency overhead caused by delivering magic states to wherever T gates are demanded by



(a) Area Scaling (b) Latency Scaling (c) Yield Rate Scaling

Figure 5.5: (a) Area required to implement a 2-level factory of varying numbers of factories X. As the distribution intensity increases, the total area increases significantly faster as factory output is scaled up. Notice that some regions are not feasible due to the constraint $K/X \leq 1$. (b) Latency as it scales with factory output capacity. For factories of a fixed capacity, increasing the number of factories on the lattice reduces latency overall and speeds up application execution time, thanks to reductions in contention and congestion. The flat tails at high K values are due to the fact that the capacity has exceeded the amount that a application ever demanded. (c) Yield as it scales with factory output capacity and number of factories. For a fixed capacity K_0, increasing the number of factories can significantly increase the success probability and yield rate of the factory.

looking at the contention and congestion factors. Finally, we will arrive at an analytical model for the overall distillation latency integrating the information from the program distribution.

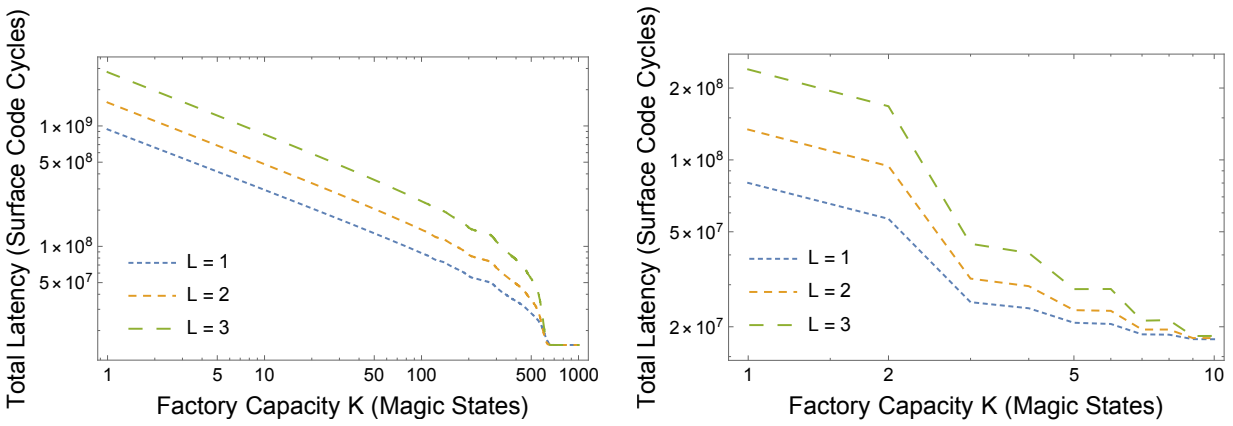
5.5.1 Program Distributions

While the majority of the prior works on this subject have been abstracting algorithm behavior into a single number, the total T gate count, we argue that the distribution of T gate throughout a algorithm has a significant impact on the performance of the magic-state factory. For example, a program with bursty T distribution, where a large number of T gates are scheduled in a few time steps, puts significant pressure on the factory's capability of producing a large amount of high fidelity magic states quickly.

In order to quantify this behavior, we choose two quantum chemistry algorithms that represent the two extremes of T gate parallelism. On one hand, the *Ground State Estimation* algorithm is an application with very low T gate parallelism. An algorithm attempting to find the ground state energy of a molecule of size m, this application can be characterized by a series of rotations on single qubits [224]. *Ising model*, on the other hand, is a highly

parallel application demanding T gates at a much higher rate. This application simulates the interaction of an Ising spin chain, and therefore requires many parallelized operations on single qubits, along with nearest neighbor qubit interactions [19]. To capture application characteristics, we use the ScaffCC compiler toolchain that supports application synthesis from high-level quantum algorithm to to physical qubit layouts and circuits [114].

The majority of the time steps in Ising Model algorithm has a large number of parallel T gates with a mean T load of 440, where as Ground State Estimation has no more than 12 T gates at each time steps. As opposed to just using the single T gate count to characterize algorithms, we will from now on use the T load distribution.



(a) Ising Model

(b) Ground State Estimation

Figure 5.6: (a)-(b) Total number of surface code cycles required by Ising Model and Ground State Estimation applications. Both figures are plotted for three different factory block-code levels, i.e. $X = 1$ and $L = 1, 2,$ and 3.

5.5.2 T-Gate Contention and Congestion

In order to fully assess the space-time volume overhead of the system, we require a low level description of how the produced magic-states are being consumed by the program.

As discussed in the Section 5.2, a T gate requires braiding between the magic-state qubit in the factory and the target qubit that the T gate operates on. Now suppose our factory is able to produce K high-fidelity magic states per distillation cycle, and at some time step the

program requests for t T gates. If we demand more than the factory could offer at once (i.e. $t > K$), then naturally only K of those requests can be served, while the others would have to wait for at least another distillation cycle. So we will say that the network has *contention* when the demand exceeds the supply capacity. By contrast, we define network *congestion* to capture the latency introduced by the fact the some braids may fail to route from the target to the factory on the 2D surface code layout, due to high braiding traffic.

To estimate the overhead of network congestion, we will perform an average case analysis without committing to a particular routing algorithm. Ideally, in the contention free limit where the number of requests t is less than K, all requests could be scheduled and executed in parallel. However, often times the requests will congest due to limitations of routing algorithms. We define a congestion *factor* $C_g$that represents the total latency required to execute all of the T gate requests at any given time.

We model congestion as a factor that scales proportional to the number of t requests made at any given time, within a particular region serviced by a factory. This assumes a general topology in which a factory is placed in the center of a region, and all of the surrounding data qubits are served by this factory alone. Naturally, the center of the region is quite dense with T gate request routes. In general, for a reasonable routing algorithm, the number of routing options increases as area available increases. However, because all of the routes have their destination in the center of the region, increasing area of the region has no such effect. In fact, the *distance* of a T request source from the factory increases the likelihood of congestion from a simple probabilistic argument. There may be other T requests blocking available routes, and the number of these possible requests that block pathways increases as the distance between a request and the factory increases. The combination of these effects interacts with the complexity of a routing algorithm, and results in a scaling relationship proportional to both the T request density t and the maximum distance of any T request

within any of these regions:

$$C_g \sim c\sqrt{t} \tag{5.14}$$

for some constant c, depending upon the routing algorithm.

We validated this congestion model in simulation using simulation tools and compiler toolchains of [112], and find that they do indeed agree. Section 5.7 discusses this in greater detail.

5.5.3 Resolving T-gate Requests

For any given program, characterized as a distribution D of the T load, we denote $D[t]$ the number of timesteps in the program that t parallel T gates are to be executed. Then the number of iterations that the factory needs to resolve the t requests can be computed based on the following latency analysis. In particular, in order to maximize the utilization of the factory, we would execute as many outstanding T gate requests as possible in parallel. When the number of requests t exceeds the factory yield K, we will need to stall the surpassed amount of requests. We denote $s = \lfloor t/K \rfloor$ the number of fully-utilized iterations. So, we are serving at full capacity for s number of times, and at each time a congestion factor is being multiplied, as discussed in Section 5.5.2. It follows that the first sK requests are completed in $s\sqrt{K}$ number of distillation cycles. And finally the rest $(t - sK)$ outstanding requests are then being executed in $\sqrt{t - sK}$ cycles. Notice that the time it takes to execute the T gate is typically shorter than the factory distillation cycle time. So under the buffer assumption made earlier, we can stage the execution of requests within a distillation cycle such that no data dependencies are violated, as long as there are magic-states available in the factory. The time required to produce some constant number k of states is T_{distill}, while the time required to deliver k states in parallel is $T_t\sqrt{k}$ due to network congestion. So the number of distillation cycles needed to supply a single cycle of k T gate requests is given by the ratio $T_{\text{t}}\sqrt{k}/T_{\text{distill}}$.

Substituting $k = K/X$ and $k = (t - sK)/X$ as described earlier, we can calculate the number of distillation iterations we need to serve t T gates in a particular timestep, as:

$$n_{\text{distill}} = \frac{T_{\text{t}}}{T_{\text{distill}}} \cdot \left(s \cdot \sqrt{\frac{K}{X}} + \sqrt{\frac{t - sK}{X}} \right) \tag{5.15}$$

where K is again the yield of each iteration from Equation 5.10.

Putting it together, we obtain our final time overhead of an application:

$$T_{\text{total}} = T_{\text{distill}} \cdot \Big(\sum_{t=0}^{T_{\text{peak}}} n_{\text{distill}} \cdot D[t] \Big) \tag{5.16}$$

where T_{peak} is the maximum number of parallel T gates scheduled at one timestep. Notice that this is independent of T_{distill}, as the distillation cycle time has been captured by the ratio T_t/T_{distill}. The scaling of this function is shown in Figure 5.5b, and is compared in Figure 5.6 across different applications.

5.6 Area and Latency Trade-offs

In this section, we will discuss some of the motivations of our proposed algorithm for optimizing space-time resource overhead, based on the area and latency analysis that we built up in the previous sections.

The Bravyi-Haah protocol shows an area *expansion* when a single factory is "divided" into many smaller factories, that is, the total area of x number of factories each with some capacity k is larger than the area of a factory with capacity $x \cdot k$. Figure 5.5a illustrates this trend, arising from the original area law equation 5.12.

Why do we want a distributed factory architecture? Although it might first seem undesirable to divide a single factory into many factories due to the area expansion, there are many advantages when doing so. One such advantage is that smaller factory can produce states with higher fidelity. So, for a fixed output capacity K, incrementing the number of

factories used to produce in total that K allows for all of those K states to have higher fidelity. The output error rate scales inversely with the number of factories on the lattice for a fixed output capacity K as seen in Equation 5.9.

This provides us with the unique ability to actually manipulate the underlying *physical error rate threshold*. In particular, substitution of K/X for K in all of the previous equations shows that the yield threshold now also has inverse dependence upon the number of factories used.

As Figure 5.4b shows, for a fixed output capacity and block code level ℓ, increasing the number of factories on the lattice can greatly increase the tolerable physical error rate under which the factory architecture can operate.

With this knowledge, we are immediately presented with architectural tradeoffs. Using the representation of programs as distributions of T gate requests, any application can be characterized by a T_{peak}, again defined as the highest number of parallel T gate requests in any timestep of an application. For a "surplus" configuration, a system may set the factory output rate $K = T_{\text{peak}}$, so as to never incur any latency during the program execution. However, as the threshold in equation 5.4 indicates, this sets an upper bound on the tolerable input error rate ϵ_{in}. With a distributed factory architecture, this provides a system parameter enabling systems to be designed that will be able to tolerate higher error rates, and still achieve the same output capacity K, at the expense of area as seen in the area law relationship from Figure 5.5a. Conversely, systems that are constructed with great knowledge of low underlying physical error rates may be able to reduce overall area of a surplus factory configuration by reducing the number of individual factories to a certain point. These are the tradeoffs in the design space that this work explores, and in fact we can find for representative benchmarks, configurations that are lower in capacity that can save orders of magnitude in space-time overhead overall.

5.7 Evaluation Methodology

5.7.1 System Configuration

Here we lay out all of the assumptions made about the underlying systems that we are studying.

First, we assume that the factories will be operated continuously. This means that each T_{distill}, the factories will produce another K_{output} states. This abstracts away the time needed to deliver these states to their destinations, which would have to be performed in a real system before the next distillation iteration begins. In such real systems, we imagine an architecture that supports a limited, fixed size buffer region so that the subsequent distillation cycle will not overwrite the previously completed states. However, this is a small constant offset in time that applies to all studied designs symmetrically, so it is omitted. Because the factories are always online and producing magic states, the overall time overhead is then equal to the number of distillation cycles required to execute all the scheduled T gate requests, multiplied by the time taken to perform a distillation iteration T_{distill} from Equation 5.8.

Next, we assume three different levels of uniformity in these designs: all distributed factories are laid out uniformly on the surface code lattice as in figure 5.3b (i.e. they are an equal distance apart), all factories in a distributed architecture are identical (i.e. they all operate with the same parameters such as K and ℓ), and within each factory each block code

Configuration	Description
Surplus	One central factory that can produce enough states to always meet the demand at each time-step of the program as in [111, 211, 156].
Singlet	One central factory that uses minimal area and produces only one state per cycle.
Optimized-Unified	One central factory that outputs an optimized number of output states per distillation cycle
Optimized-Distributed	A optimized set of factories that together output an optimized number of output states

Table 5.3: List of architecture configurations explored in this work.

round is identical (i.e. they are composed of identical $n \to k$ protocols). Note that Campbell et. al. in [156] allows varying k within a single factory, across different rounds.

In performing our evaluations we consider four different system configurations: *surplus* architectures that minimize application latency by setting the magic-state output capacity to the peak T gate request count in an application, *singlet* architectures that minimize required space for the factory by producing only a single state per distillation cycle, *optimized-unified* architectures that use one central factory with an optimized choice of output capacity K and number of distillation rounds ℓ, and *optimized-distributed* architectures that choose an optimum output capacity K distributed into an optimum number X of factories, each utilizing ℓ distillation rounds. These architectures are summarized in Table 5.3.

5.7.2 Optimization Algorithm

As keen readers may have already observed from Figure 5.6 and Figure 5.4d, for fixed output capacity K, it costs us both in time latency and in factory footprint to implement a high ℓ block-code factory. The only reason we design for high ℓ is to achieve the desired target error rate. This relation is best captured in the bottom half of Figure 5.7, where the $L = 1$ factory is not feasible for $K \geq 1$ since its output error rate is higher than the target error rate, while the $L = 2$ factory is feasible for $K \in [1, 50]$, and the $L = 3$ factory is feasible for the entire plotted range.

We combine all of the details of the explicit overhead estimation derived above in order to find optimal design points in the system configuration space. To do this, we must ensure that designs are capable of producing the target logical error rate for an application. Additionally, there exists a set of constraints C that $K, X \in \mathbb{Z}^+$ have to satisfy: (i) $1 \leq X \leq K$; (ii) $K/X \leq (1 - 8\epsilon_{\text{inject}})/(3\epsilon_{\text{inject}})$, due the Bravyi-Haah protocol error thresholds. With the feasible space mapped out, standard nonlinear optimization techniques are employed to explore the space and select the space-time optimal design point.

With these constraints in mind, we explore the space by first selecting the lowest ℓ possible.

As the area law and full volume scaling trends of the previous sections indicate, if there are any feasible design points with $\ell = \ell_0$, then any feasible design points for systems with $\ell_i > \ell_0$ will be strictly greater in overall volume. This is somewhat intuitive, as concatenation of block code protocols is very costly.

With the lowest ℓ selected, we check to see if there exists any feasible design points for this ℓ by checking for solutions to the equation:

$$(1 + 3k^{1/\ell})^{2^\ell - 1} \epsilon_{\text{inject}}^{2^\ell} \leq \frac{P_s}{N_{\text{gates}}} \tag{5.17}$$

If the K that solves this equation is greater than or equal to 1, then there does exist feasible design space along this ℓ, and the algorithm continues. Otherwise, ℓ is incremented.

Next, nonlinear optimization techniques are used to search within the mapped feasible space for optimal design points in both K and X.

5.7.3 Simulation and Validation

This section explores the validity of our models through empirical evaluation of the space-time resources. To do this, we improve the surface code simulation tool from [112] to accurately assess the latency and qubit cost of fully error-corrected applications with various magic-state distillation factory configurations. Specifically, we added support for arbitrary factory layouts, which manifests as black boxed regions dedicated to factories that cannot be routed through during computation, combined with sets of locations of produced magic states. The result is a cycle precise simulator that accurately performs production and consumption of magic states, including all necessary routing.

One implementation detail that is supported is the ability to dynamically reallocate specific magic-state assignments during runtime. Statically, each T gate operation is prespecified with a particular magic-state resource, located along the outer edge of a factory. During runtime, this can introduce unnecessary contention, as two nearby logical qubits can potentially request

the same magic state. This is avoided by implementing online magic-state resource shuffling, so that if the particular state that was requested is unavailable, the system selects the next nearest state that is available. If no such states exist, this T gate is stalled until the next distillation cycle is completed.

Figure 5.8 shows simulation results superimposed on top of those driven analytically. We can see that the model shows the same trend as the simulation behavior (blue line), and thus we will be able to show relative tradeoffs between capacity and latency. For simplicity the validation is performed on single unified factory located at the center of the surface code lattice. The results extend well to multiple factories, because in the distributed case, each factory will be responsible for magic-state requests in a sub-region of the lattice.

We can validate this by simulating optimal operating points in the space-time trade-off spectrum and comparing them to our expectation from the model. Using simulation data, we re-plot our idealized tradeoff in Figure 5.1 for the Ising Model Application and show the results in Figure 5.9. We see that as factory capacities increase, the applications time improves at the expense of its qubit numbers. In this figure, the space-time volume is sketched in green, and has two near-optimal points: one with relatively few qubits but high latency, and vice versa. The worst performance occurs in the middle of this spectrum, when transition from level 1 to level 2 distillation needs to occur (causing a sudden jump in qubits, but not much latency improvement).

5.8 Results

In this section we present the resource requirements of various magic-state factory architectures, and show that by considering the scaling behaviors that we have highlighted and searching the design space with our optimization algorithm, we can discover system configurations that save orders of magnitude of quantum volume.

We first compare the overheads of the surplus and singlet architectures that represent baselines against which we compare our optimized architectures. We then compare the surplus

architecture with the optimized-distributed design found with our optimization algorithm. We look at two representative benchmarks for the quantum chemistry and quantum simulation fields, the Ising Model [19] and Ground State Estimation [224] algorithms, as well as how performance of these architectures changes as the benchmarks scale up in size. Next, we detail the space and time trade-off that is made in our resource optimized design choices, and show that the latency induced by a design is a more dominant factor in these applications. We then present a full design space comparison, showing the performance of the surplus design against the singlet design, as well as the optimized-unified factory design, all compared to the performance of optimized-distributed design. Lastly, we analyze the sensitivity of these designs to fluctuations in the underlying physical error rates, and show that building out a distributed factory design adds robustness that makes the architecture perform well for a wider range of input parameters.

5.8.1 Comparing Surplus and Singlet Architectures

We begin with Figure 5.10 by comparing two architectures that aim solely to minimize application latency or required space. This comparison represents the range between two ends of the design space spectrum for single factory architectures, and each shows a particular error rate range over which it performs more optimally. Initially, at the highest input error rate, the space optimal singlet design requires more resources than the time optimal surplus design, as the application suffers from excessive latency from magic-state factory access time. Note the inflection points at $10^{-3.5}$ and $10^{-4.5}$ input error rates. At these points, the singlet factory is able to reduce the number of rounds of distillation it must perform, as input error rates are sufficiently low. Over this region, the reduction in area compensates the expansion in computation time, and the design outperforms the much larger surplus factory configuration. At $10^{-4.5}$, the surplus factory is able to operate with fewer distillation rounds as well, enabling this configuration to outperform the singlet design.

This behavior is surprising, as it indicates that with respect to a high-parallelism appli-

cation, there are input error rate regions where intuitively conservative, space minimizing designs are able to outperform what seem like aggressively optimized designs. We see this because we are comparing space and time simultaneously, which allows us to see that the trade-off is asymmetric and these factors interact non-trivially.

5.8.2 Optimized Design Performance

We now move to comparing the surplus design against the optimized-distributed design discovered by our optimization algorithm, that is allowed to subdivide factories across the machine. Figures 5.11a and 5.11b depict the detailed results of our optimization procedure on the Ising Model and Ground State Estimation applications, respectively. Ising Model is intrinsically very parallel, which leads to a higher optimal capacity choice for the optimized-distributed factory. Note however that it is able to choose a distribution level that saves approximately 15x in space-time volume. Ground State Estimation is very serial, yet for sufficiently low error rates the optimized-distributed design is able to incorporate distribution of factories into the lattice to lower the required block code concatenation level ℓ, resulting in a 12x reduction in volume across these points.

The reason that the distributed factory design is able to outperform the surplus design is that the feasibility regions of the two designs differ. Because the distributed factory utilizes many small factories on the machine it can achieve a higher output state fidelity than a single factory design, which enables it to operate with a smaller number of distillation rounds. The optimization algorithm respects this characteristic, which is why it searches iteratively from the lowest number of distillation rounds possible, one by one until it discovers a feasible factory configuration.

Optimized Design Performance Scaling

Figures 5.11c and 5.11d detail these trends as larger and larger quantum simulation applications are executed. For extremely large simulations, we find that the volume reductions that

optimizing a factory design yields become even more pronounced, resulting in between a 15x and 18x full resource reduction. These designs also show sensitivity to physical error rates that require designs to change block code distillation level.

5.8.3 Distributed Factory Characteristics

As Figure 5.12a describes, an optimized-distributed set of factories is able to save between 1.2x and 4x in total space-time volume over the optimized-unified factory. Large volume jumps occur primarily between $10^{-3.5}$ and $10^{-3.4}$ physical error rate, and this again corresponds to a requirement by this application to increment to a higher block code level ℓ, which happens for both the unified and distributed factory schemes.

These optimized designs trade space for time, as Figures 5.12a and 5.12b indicate, and the net effect is an overall volume reduction. This is indicative that for these highly parallel quantum chemistry applications, the magic-state factory access latency is a much more dominating effect than the number of physical qubits required to run these factories.

Figure 5.12c depicts the output capacities chosen by the optimization procedure, and how they differ when the system is unified or distributed. Notably, at both ends of the input error rate spectrum we find that both factory architectures choose the same output capacity, as in the high error rate case this is driven by high ℓ requirement, while in the low error rate limit both factory architectures can afford to be very large and not suffer from any yield penalties. However, through the center of the error rate spectrum the unified factory design must lower the chosen output capacity, as supporting higher capacity would require a very expensive increase in the number of distillation rounds.

5.8.4 Full Design Space Comparison

Figure 5.14 depicts the full space-time volume required by different factory architectures across the design space. Shown are the four main configurations: a surplus factory configured with output capacity $K = T_{\text{peak}}$, a singlet factory with $K = 1$, an optimized-unified factory,

and an optimized-distributed factory.

Distinct volume phases are evident visually on the graph, due to the different feasibility regions of the architectures. Sweeping from high error rates to low error rates, large volume jumps occur as observed before, for specific configurations when that configuration can operate with fewer rounds of distillation in order to convert the input error rate to the target output error rate. Notice that this jump occurs earliest for the singlet, optimized-unified, and optimized-distributed designs, at $10^{-3.5}$ input error rate. All of these designs show an inflection point here, where the configurations can achieve the target output error rate with a smaller number of block code distillation levels. This is not true of the surplus factory, which in fact has the largest output capacity of the set. Because the output capacity is so high, the lowest achievable output error rate is much higher than that of the other designs. This forces the block code level to remain high until the input error rate becomes sufficiently low, which occurs at $10^{-4.5}$.

5.8.5 Sensitivity Analysis

Now we turn to analyzing how these designs perform if the environment in which they were designed changes. Supposing that a design choice has been made specifying the desired factory capacity K, number of factories X, and block code distillation level ℓ, different types of architectures show varying sensitivity to fluctuations in the underlying design points around which the architectures were constructed. For example, Figure 5.13 details an instance of this occurrence. The figure shows the surplus, singlet, and optimized-distributed factory designs, in this case setting $K \sim 600$ and $X \sim 200$ for the distributed architecture. All of these factories were designed under the assumption that the physical machine will operate with 10^{-5} error rate.

We see that while these applications perform similarly over the range from 10^{-5} to 10^{-4}, just after this point the surplus factory encounters a steep volume expansion due to the yield

threshold equation 5.11. For this design the threshold of tolerable physical error rates is quite high, significantly higher than that of the other designs. Because of this, it can tolerate a smaller range of fluctuation in the underlying error rate before it ceases to execute algorithms correctly.

5.9 Conclusion

We present methods for designing magic-state distillation factory architectures that are optimized to execute applications that present with a specific parallelism distribution. By considering applications with different levels of parallelism, we design architectures to take advantage of these characteristics and execute the application with minimal space and execution time overhead.

By carefully analyzing the interaction between various magic-state factory characteristics, we find that choosing the most resource optimized magic-state distribution architecture is a complex procedure. We derive and present these trade offs, and compare the architectures that have been commonly described in literature. These comparisons show a surprising picture: namely that even a modest factory capable of producing just a single resource state per distillation cycle can outperform the more commonly described surplus factory in particular input error rate regimes. We also propose a method of distributing the total number of magic states to be produced into several smaller factories uniformly distributed on a machine. In doing this, we see that these types of architectures are capable of achieving higher output fidelities of their produced states with added resilience against fluctuations of the underlying error rate, when compared to unified architectures composed of a single factory. While these designs are tailored to specific applications, we conjecture that distributed systems would in fact be more flexible in their abilities to execute applications with different amounts of parallelism. Intrinsic to their design is the ability to optionally compile smaller applications to various subunits of the machine. Because of this, these designs can be used to support a much wider range of application types than those comprised of a single factory.

These systems also show that the trade off in space and time is asymmetric. In quantum chemistry and simulation applications, we notice that the resource optimized designs can use upwards of 2 orders of magnitude more physical qubits to be implemented, while they end up saving over 3 orders of magnitude in time. Magic-state access time, or latency induced specifically by delays due to stalling as magic states are produced, we find is a dominating effect in the execution of these applications. In order to mitigate these effects in a resource-aware fashion, designing a distributed system of several factories allows for efficient partitioning of the magic-state demand across the machine, at the cost of physical area.

These conclusions can have physical impacts on near-term designs as well. Specifically, the construction of a factory architecture can imply the location of physical control signals on an underlying device. What we are showing then is the effect of several theoretical long-term designs, and the conclusion that distributed sets of factories outperform other designs should help motivate device fabrication teams as they decide which physical locations should be occupied by rotation generating control signals. As a general principle, long term architectural design and analysis can help guide the study and development of near term devices, which ultimately will help hasten the onset of the fault-tolerant era [178].

5.10 Future Work

There are a number of immediate extensions to this study:

- *Comparing distributed factory topologies.* Choosing an optimal layout for a distributed factory design is potentially very difficult, and requires an ability to estimate the overheads associated with different layouts. Using architectural simulation tools and adapted network simulation mechanisms, we can foresee evaluation of two new architectures: peripheral and asymmetric-mesh placement. Peripheral placement refers to factories surrounding a central computational region, while asymmetric-mesh placement refers to embedding the factories throughout the machine itself.

- *Embedding data qubits within magic-state factories.* While the designs presented here assume that magic-state factory regions are to be considered black boxes that are not to be occupied by data qubits, because of their massive size requirements we imagine a system that embeds the relatively smaller number of data qubits within the factories themselves. A study of the effect of various embedding techniques on factory cycle latency could determine the efficiency of such a design.

- *Advanced factory pipeline hierarchy.* We envision a concatenation of clusters of the magic-state factories, targeting continuous outputs in time, and hence reduction in contention caused by the distillation latency. In particular, each sub-region in the mesh contains multiple small, identical factories that were turned on asynchronously. So at each time step, there will always be a factory that completes a distillation cycle, and thus serving magic state continuously.

- *Generalization to other distillation protocols.* Although the Bravyi-Haah protocol studied in this paper is among the best known protocols, little analysis has been done on other techniques discovered recently [93].

- *Optimizing the internal mapping and scheduling of magic-state factories.* This work has modeled factories as black-boxed regions that continuously produce resources. A realistic implementation of those factories that optimize for internal congestion would significantly reduce factory overhead, in conjunction with designs proposed in this work that optimize for external congestion. This was studied in [54].

- *Flexibility of Distributed Magic-State Architectures.* While these designs are tailored to applications of a certain parallelism distribution, a study could analyze designs that balance domain specific optimization against general application compatibility.

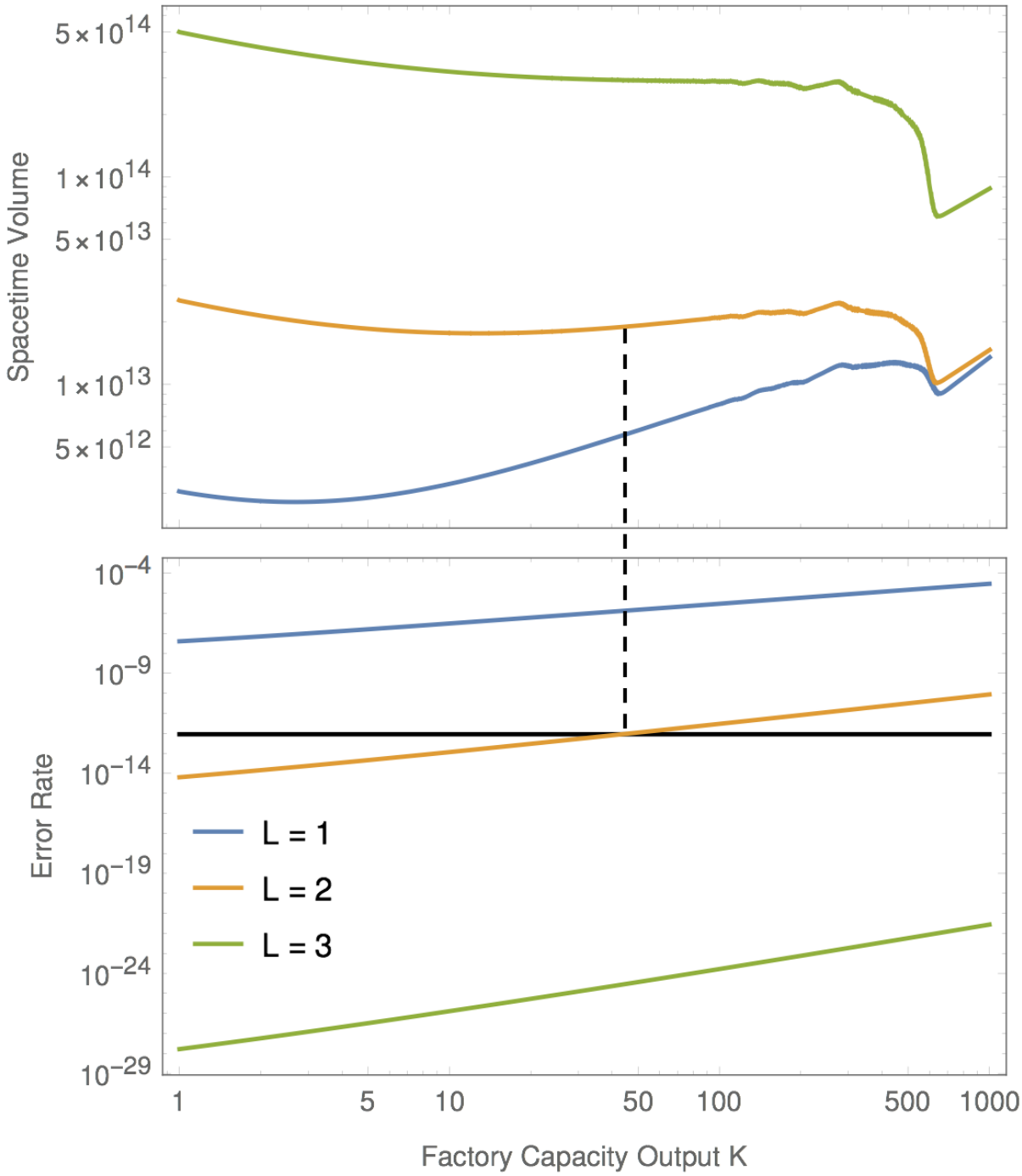


Figure 5.7: Space-time volume minimization under error threshold constraints imposed by target error rate for each block code level. An application will set a target error rate (black) that the factory must be able to achieve in output state fidelity. On the lower plot, levels 2 and 3 are the only levels available that can satisfy this. In the upper plot, we find that the lowest volume in the feasible area is located on the level 2 factory feasibility line. Recall the volume shapes are explained earlier in section 5.5. Here the tails after $K \approx 800$ show an increase in volume, as the added capacity grows the factory areas while maintaining constant latency.

Algorithm 1 Space-time Optimization Procedure

Require: P_s, N_{gates}, ϵ_{inject}, distribution D and constraints C
Ensure: K, X

procedure OPTIMIZE
  $K \leftarrow 1$, $X \leftarrow 1$, $\ell_{\max} = 5$
  $\epsilon_{\text{target}} \leftarrow P_s / N_{\text{gates}}$
  for $\ell \in [1, \ell_{\max}]$ **do**
    $k_\ell \leftarrow (K/X)^{1/\ell}$
    $n_\ell \leftarrow 3k_\ell + 8$
    for $r \in \{1, \cdots \ell\}$ **do**
      if $r == \ell$ **then** $\epsilon_r \leftarrow \epsilon_{\text{target}}$
      else $\epsilon_r \leftarrow (1 + 3k_\ell)^{2^r - 1} \epsilon_{\text{inject}}^{2^r}$
      end if
      $d_r \leftarrow \text{Solve}\{d_r \cdot (100\epsilon_{\text{in}})^{(d_r+1)/2} = \epsilon_r,\ d_r\}$
    end for
    $R \equiv K/X \leftarrow \text{Solve}\{\epsilon_\ell = \epsilon_{\text{target}},\ R\}$
    if $R \geq 1$ **then**
      $K_{\text{output}} \leftarrow K \cdot \prod_{r=1}^{\ell} \left[(1 - n_\ell \cdot \epsilon_{\text{inject}})\epsilon_r\right]$
      $s \leftarrow \lfloor t / K_{\text{output}} \rfloor$
      $T_{\text{t}} \leftarrow 4d_\ell + 4$
      $T_{\text{distill}} \leftarrow 11 \sum_{r=1}^{\ell} d_r$
      $n_{\text{distill}} \leftarrow \frac{T_{\text{t}}}{T_{\text{distill}}} \cdot \left(s \cdot \sqrt{\frac{K}{X}} + \sqrt{\frac{t - sK}{X}}\right)$
      $T_{\text{total}} \leftarrow T_{\text{distill}} \left(\sum_{t=0}^{T_{\text{peak}}} n_{\text{distill}}\right) \cdot D[t]$
      $A_{\text{factories}} \leftarrow X \cdot n_l^{\ell - 1} \cdot (6k_\ell + 14) \cdot d_1^2$
      $(K, X) \leftarrow \arg\min_{(K,X):C} A_{\text{factories}} \cdot T_{\text{total}}$
    else
      $\ell \leftarrow \ell + 1$
    end if
  end for
  return K, X
end procedure

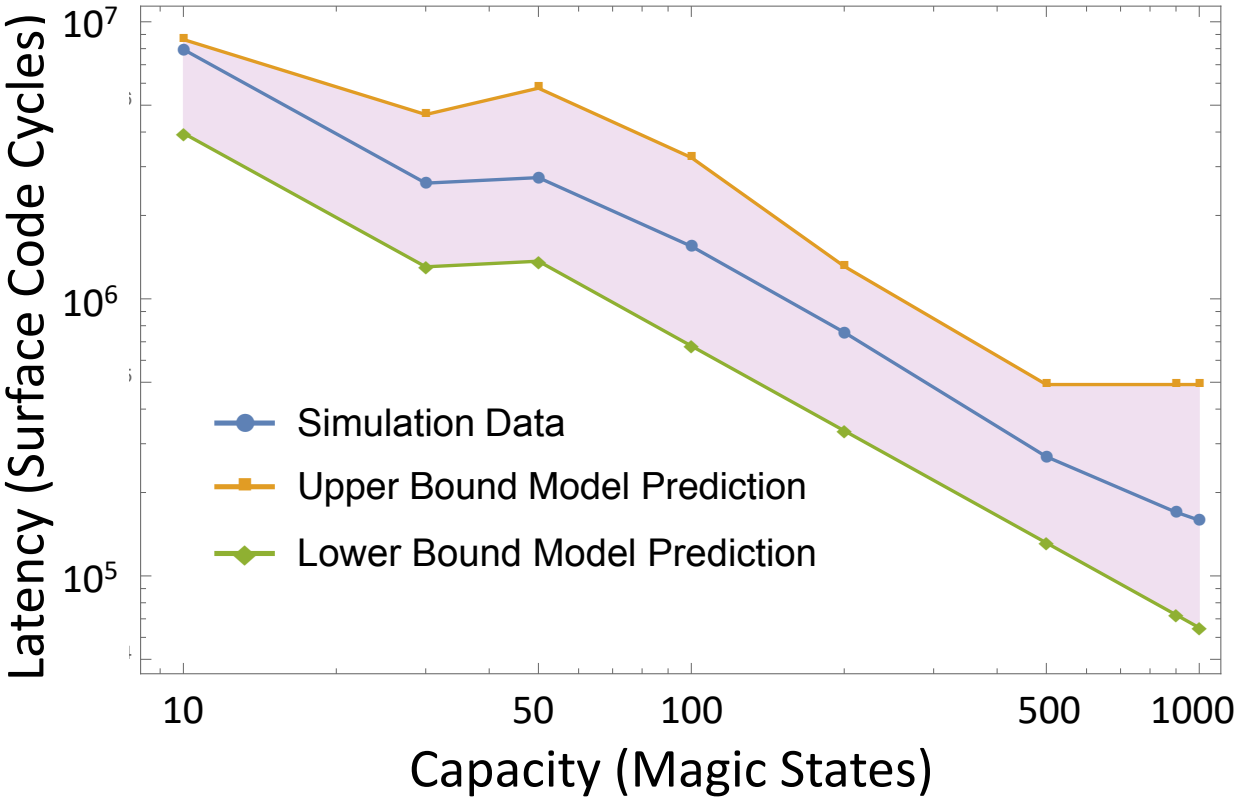


Figure 5.8: Model validation by simulation. The simulation data (blue line) lies between the upper bound model prediction that overestimates congestion(orange line), and the congestion-free lower bound (green line).

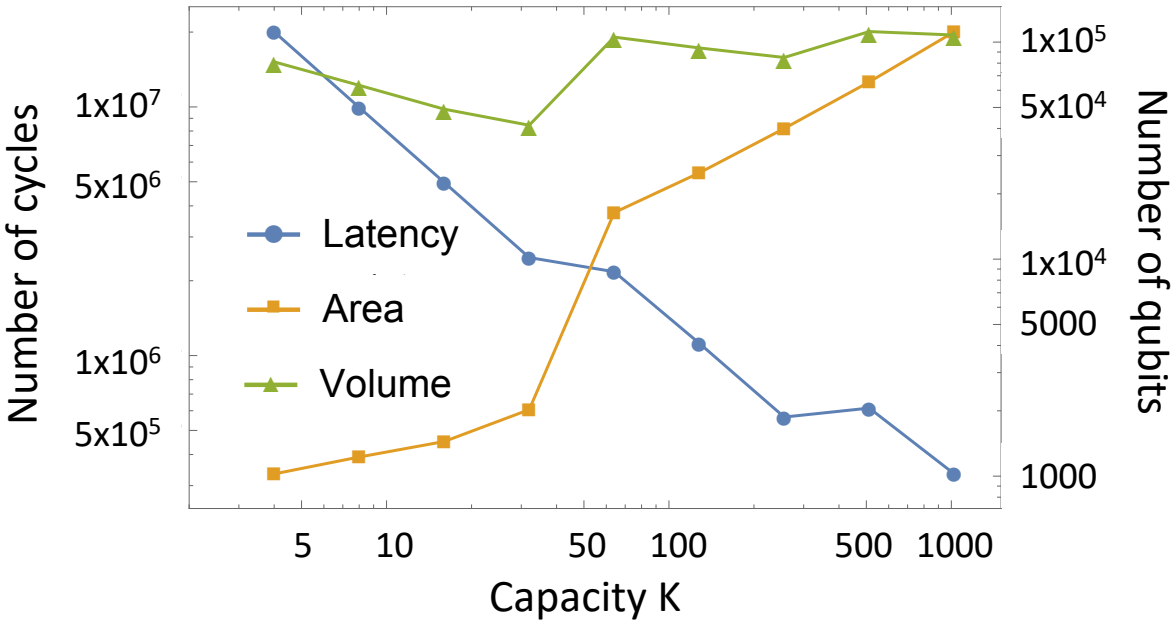


Figure 5.9: Space-time tradeoff observed empirically in simulation for varying factory capacities. A space-time volume (green line) can be chosen at $K \approx 40$, which is an optimal, minimized value on this curve. It corresponds here to a low-qubit, high latency configuration. Notice that another configuration at $K \approx 300$) could be chosen, corresponding to a high-qubit, low-latency configuration. In this case, the former of these choices is more resource optimized, as the space-time cost is lower.

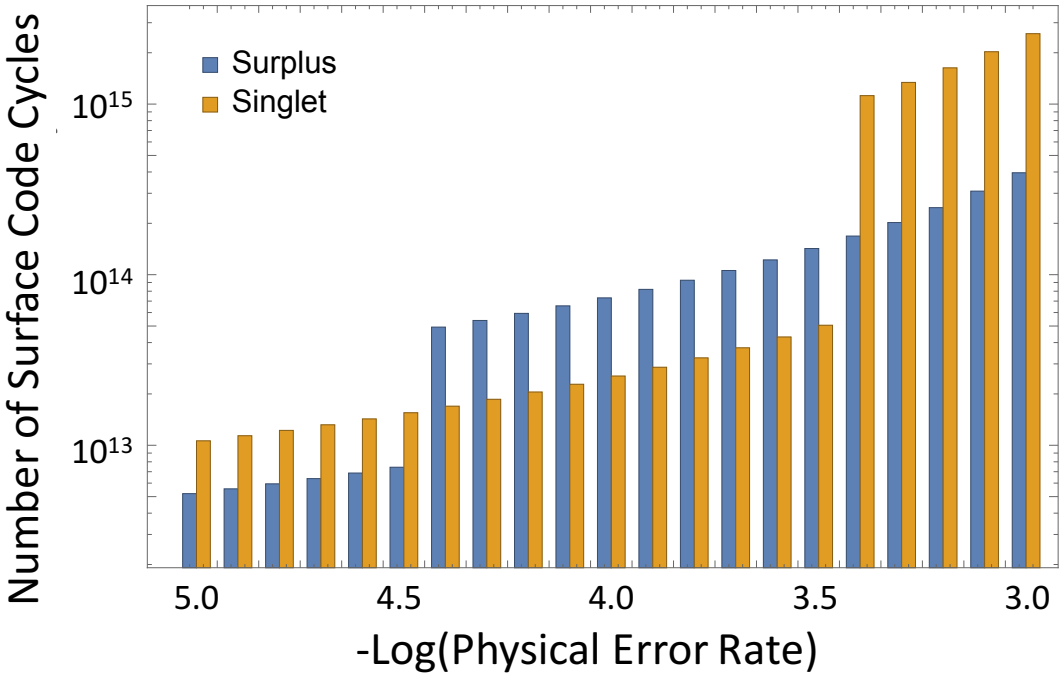
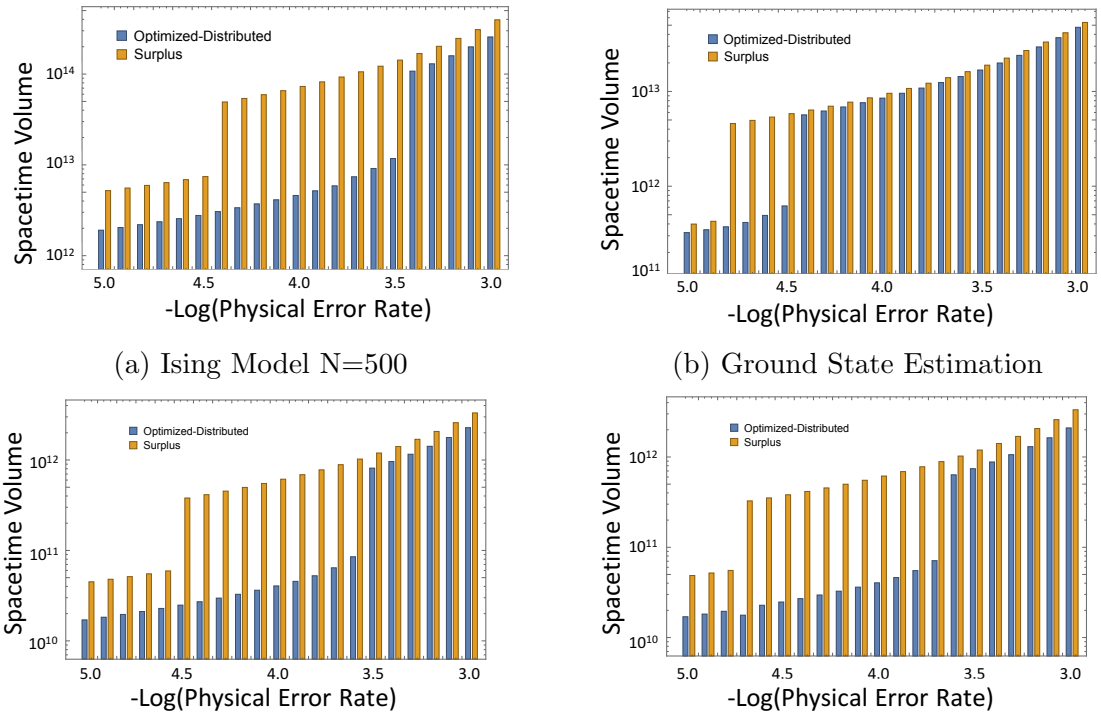


Figure 5.10: (Color online) Comparing surplus and singlet designs. There are regions where each outperforms the other, showing great sensitivity to the underlying physical error rate and the corresponding required ℓ. Recall that the step-like shape is due to level transitions explained in section 5.7.2.



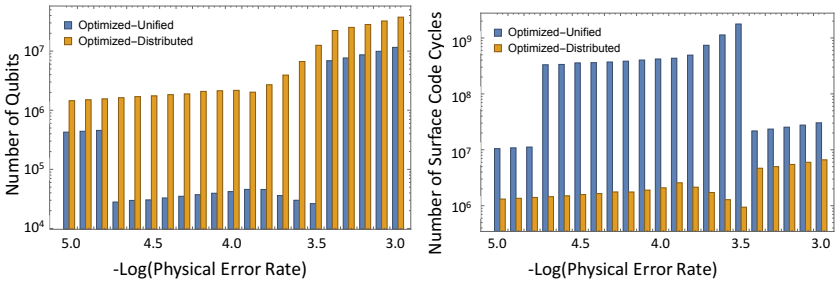
(a) Ising Model N=500

(b) Ground State Estimation

(c) Ising Model N=1000

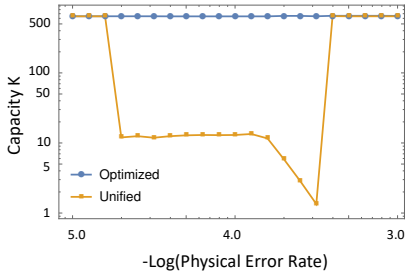
(d) Ising Model N=2000

Figure 5.11: (Color online) (a)-(b) Resource reductions of optimized-distributed designs over surplus designs for both Ising Model and Ground State Estimation. While Ising Model is intrinsically more parallel which leads to high choices of output capacity, both applications still show between a 12x and 16x reduction in overall space-time volume. (c)-(d) Ising Model with varying problem sizes, comparing time optimal factories against fully space-time optimized configurations. We see that the trend of between 15x and 20x total volume reduction extends to larger molecular simulations.



(a) Space tradeoff

(b) Time tradeoff



(c) Output capacities procedurally selected

Figure 5.12: (Color online) Space-time volume reduces by moving from an optimized-unified factory to an optimized-distributed factory, as the designs trade space for time. Magic-state access latency is a dominating effect in these applications, as can be seen by the large capacity values chosen by the optimized factory configuration.

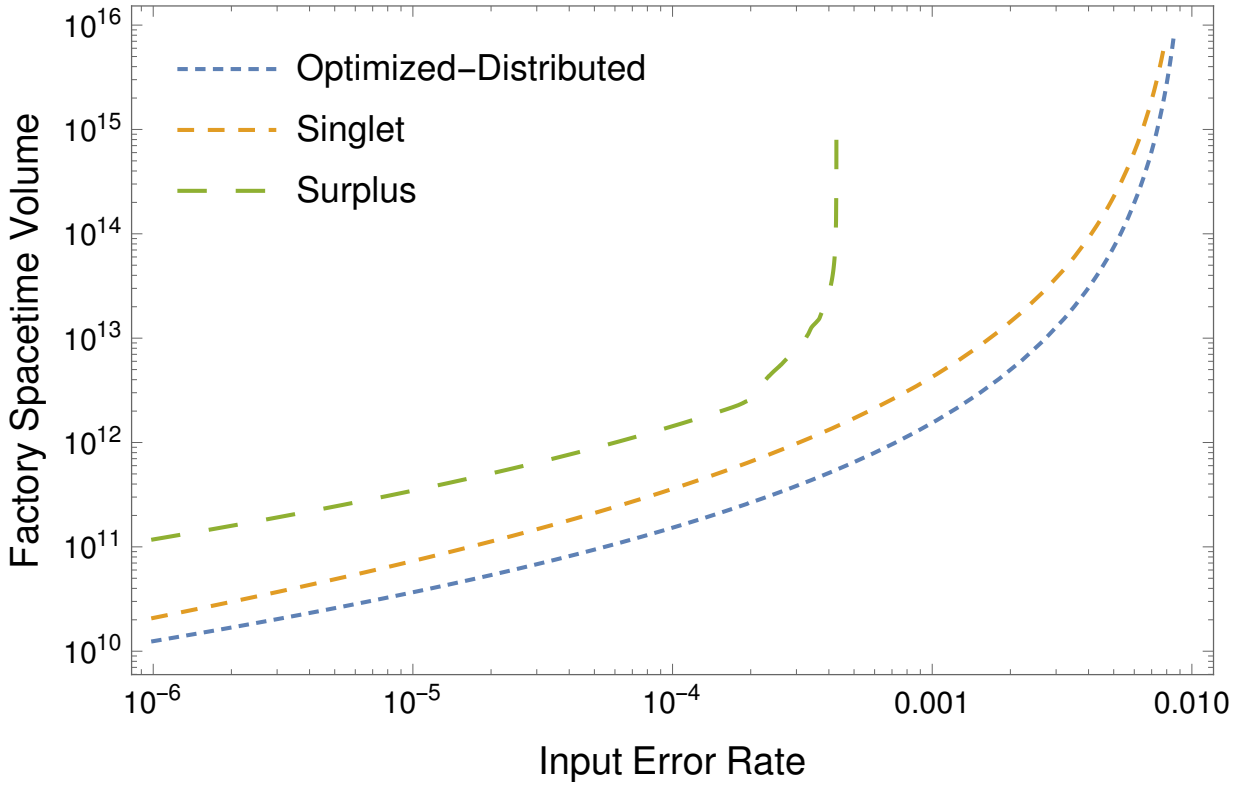


Figure 5.13: Factory architectures and their sensitivities to fluctuations in underlying physical error rates

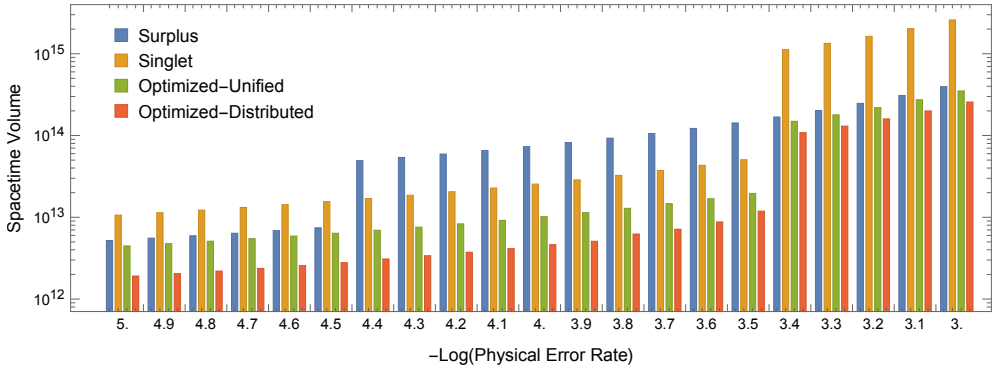


Figure 5.14: (Color online) Full volume comparison across distillation factory architectures.

Part III

Near-Term Quantum Algorithm Design

CHAPTER 6

IMPACTS OF QUBIT CONNECTIVITY ON QUANTUM ALGORITHM PERFORMANCE

Quantum computing hardware is undergoing rapid development from proof-of-principle devices to scalable machines that could eventually challenge classical supercomputers on specific tasks. On platforms with local connectivity, the transition from one- to two-dimensional arrays of qubits is seen as a natural technological step to increase the density of computing power and to reduce the routing cost of limited connectivity. Here we map and schedule representative algorithmic workloads - the Quantum Fourier Transform (QFT) relevant to factoring, the Grover diffusion operator relevant to quantum search, and Jordan-Wigner parity rotations relevant to simulations of quantum chemistry and materials science - to qubit arrays with varying connectivity. In particular we investigate the impact of restricting the ideal all-to-all connectivity to a square grid, a ladder and a linear array of qubits. Our schedule for the QFT on a ladder results in running time close to that of a system with all-to-all connectivity. Our results suggest that some common quantum algorithm primitives can be optimized to have execution times on systems with limited connectivities, such as a ladder and linear array, that are competitive with systems that have all-to-all connectivity."

6.1 Introduction

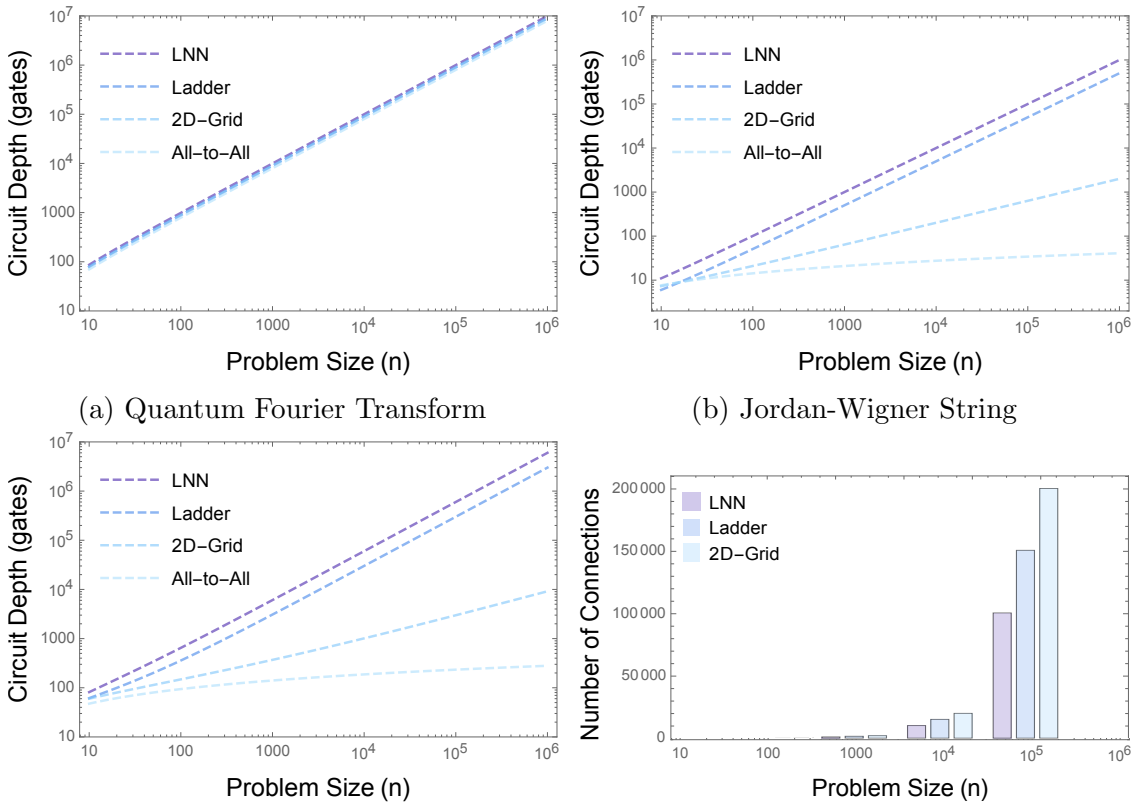
Quantum devices may one day be able to perform computational tasks that exceed the capabilities of classical computers. To this end, quantum algorithms as well as quantum hardware are active areas of research. Given the challenging nature of this goal, co-design between quantum hardware and quantum algorithms will be integral to constructing useful quantum computers. In this paper we present research addressing how the physical qubit layout affects its suitability for a specific application area.

Quantum algorithms are currently developed using high-level abstractions in which

quantum circuits are described in terms of an ideal qubit register in which two- or multi-qubit gates can be performed between any subset of qubits. In practice though, almost all physical architectures will allow for two-qubit operations between a limited set of qubit pairs, a property that can be termed as the connectivity of the qubits. Additional routing operations or teleportation protocols are then required to implement the desired two-qubit gates, and these solutions may introduce a substantial overhead in terms of the execution time of the algorithm or in the number of ancilla qubits. This makes highly-connected devices, where two-qubit gates can be performed between as many pairs of qubits as possible, seemingly favorable platforms. However, higher connectivity devices are harder to manufacture and the difficulty is experienced across multiple technologies and architectures ranging from ion traps [135], to superconducting qubits [18, 218], to quantum dots [139]. The impact of technological limitations can be reduced by developing compilation tools and scheduling techniques that soon will play an essential role in the operation of real quantum computers. Since a majority of quantum algorithms leverage a relatively small set of algorithmic primitives, optimizing the scheduling of these to physically feasible connectivity graphs is advantageous and is the goal of this paper.

For Noisy Intermediate Scale Quantum (NISQ) computers [179], composed of 50-100 physical qubits which function with no or only partial quantum error correction, having optimized schedules determines whether the results from running the program are significant or are too noisy to be meaningful. The ultimate objective is to have error-corrected qubits, each of which consists of many physical qubits, which may be operated for (in-principle) infinitely long times. The encoded qubits will likely also have a limited connectivity graph. Therefore, the research in this paper remains relevant also in the long term by contributing to the very practical task of reducing the execution time of quantum algorithms.

Other related works in this area have focused on optimizing specific instances of workloads at the application level for specific physical device types [71, 58, 130], or for scheduling specific applications onto linearly connected devices [92, 195, 187, 98, 127, 227]. A smaller



(a) Quantum Fourier Transform

(b) Jordan-Wigner String

(c) Grover's Diffusion Operator

(d) Connection Complexity Scaling

Figure 6.1: Optimized algorithm performance (a) Quantum Fourier Transform, (b) Jordan-Wigner String, (c) Grover's Diffusion Operator, with each precise running time analyzed on various hardware architectures. (d) shows the scaling of hardware connections for each architecture. All-to-all machine performance is shown, but the number of connections is omitted as it is significantly greater than the other machines.

body of work has been developed applying similar scheduling and optimization techniques to two-dimensional qubit plane arrays [185, 196, 38]. Additionally, other work has been developed from the perspective of adapting applications to specific physical devices by leveraging techniques from optimal control theory [192].

In this work we analyze the impact of three connectivity graphs, namely a line, ladder, and two-dimensional square grid, on the scaling of important algorithmic workloads like the quantum Fourier transform, Grover diffusion operator, and the parity-based rotations arising from mapping electrons to qubits using the Jordan-Wigner transform. Our results have been obtained by hand-optimizing schedules for the quantum algorithms in the paper on different connectivity graphs. We observe that common quantum algorithmic primitives

exhibit structure that can be exploited to produce efficient schedules. Despite it being known that linear connectivity graphs impose an overhead that is at most polynomial in the number of qubits and that does not jeopardize quantum speedup, in some cases we observe surprisingly that adding connectivity provides marginal performance benefits after optimizing the circuits.

A high-level summary of our main results is presented in the next section, Section 6.2. Section 6.3 introduces the qubit connectivity graphs considered in this study and Section 7.7 goes over our results in detail. Noisy simulations of each algorithm is provided and analyzed in Section 6.5. We conclude with Section IV. Since our topic is inter-disciplinary, straddling the boundary between physics and computer science, we have provided an introduction to the notation and algorithms we use. Readers unfamiliar with quantum computation can find detailed background and notation information, as well as details about the algorithms chosen, in the Appendix sections 7.3 and 6.8. Derivations of gate decompositions used for gate count normalization is included in Appendix 6.9.1 and 6.9.2. Explicit formulations of all algorithms that were omitted from the main text for brevity are contained in Appendix 6.9.3, and an analysis of the gate counts of the algorithms allowing for arbitrary two-qubit operations is included in Appendix 6.9.4.

6.2 Summary of Results

Circuit Depths

Architecture	QFT	JW String[5]	Grover's Diffusion Operator $n-1$ Ancilla	Grover's Diffusion Operator 1 Ancilla [17]
All-to-All	$8n - 10$[2]	$2\lceil \log_2 n \rceil + 1$	$14 \log_2 n + 1$	$48n - 204$
Linear Nearest Neighbor	$10n - 13$	$n + 1 + (n \bmod 2)$	$6n + 8 \log_2(n) - 5$	$672n - 2928$
Ladder	$9n - 11$	$n/2 + 1 + (n/2 \bmod 2)$	$3n + 8 \log_2(n) + 13$	$48n - 204$
2D Grid	$10n - 13_1$	$2\sqrt{n} + 1 + 2(\sqrt{n} \bmod 2)$	$9\sqrt{n} + 8 \log_2(n) + 13$	$48n - 204$

Table 6.1: Circuit depth results after gate decomposition. Here n is the number of qubits involved in each algorithm. For the Grover's Diffusion Operator algorithms, it excludes any necessary ancilla which are explicitly stated in column headers. Standard gate set used for analysis includes all single qubit rotations (including Hadamard) and CNOT operations.

Table 6.1 shows the main results of our paper. It outlines the circuit depth under the assumptions of the gate set made in the next section. Additionally, Table 6.2 shows the total

Total Gate Counts

Architecture	QFT	JW String	Grover's Diffusion Operator $n-1$ Ancilla	Grover's Diffusion Operator 1 Ancilla
All-to-All	$n(2n-1)$	$2n-1$	$14(n-1)+1$	$48n-204$
Linear Nearest Neighbor	$n(5n-3)/2$	$2n-1$	$6n\log_2(n)+2n-1$	$672n-2928$
Ladder	$n(9n/2-3)/2$	$2n-1$	$3n\log_2(n)+12n-1$	$48n-204$
2D Grid	$n(5n-3)/2^2$	$2n-1$	$(3n+3/2\sqrt{n})\log_2(n)+2n+11\sqrt{n}+5$	$48n-204$

Table 6.2: Total gate count results after gate decomposition. n is again the number of qubits involved in each algorithm. For the Grover's Diffusion Operator algorithms, it excludes any necessary ancilla which are explicitly stated in column headers. Standard gate set used for analysis includes all single qubit rotations (including Hadamard) and CNOT operations.

gate counts required for each algorithm. The main results are:

Quantum Fourier Transform: Though the quantum circuit involves gates between every possible pair of qubits, the circuit depth is linear in the number of qubits even on a line, which is the same scaling as on an all-to-all connectivity graph. Adding $n/2-1$ connections to form ladder connectivity enables optimization that makes the circuit depth very close to optimal.

Jordan-Wigner String: The Jordan-Wigner string is a building block of fermionic simulation algorithms on quantum computers. On classical computers, these simulations take $\mathcal{O}(\exp(n))$ time, where n is the number of qubits or electronic orbitals. On quantum computers, the fermionic simulation algorithms involve a polynomial number of Jordan Wigner strings. The run-time for a single Jordan-Wigner string on qubits with all-to-all connectivity is $\mathcal{O}(\log(n))$. Constraining the connectivity increases the run-time exponentially to become polynomial in n. The run-time on a ladder is half that on the line, and the 2D grid is quadratically faster than both of these.

Grover's Diffusion Operator: In quantum search, the Grover diffusion operator is called after every application of the oracle. For a search space of $N=2^n$, a classical search algorithm calls the oracle $\mathcal{O}(N)$ times, whereas the quantum algorithm calls the oracle $\mathcal{O}(\sqrt{N})$ times. The run-time for the Grover diffusion operator on an all-to-all connectivity qubit device is $\mathcal{O}(\log(n))$ which increases exponentially to be polynomial in n on a constrained connectivity device. Here too, going to a ladder from a line halves the runtime, and the 2D grid is

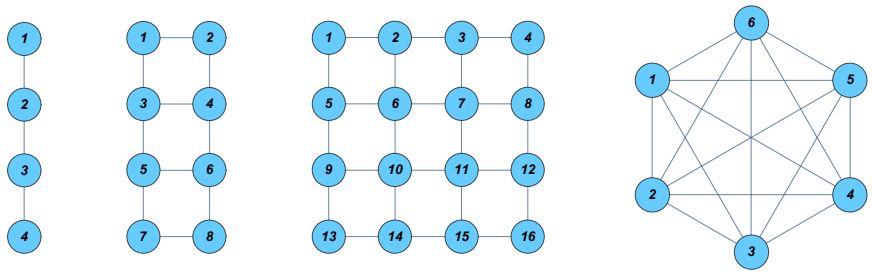


Figure 6.2: Connectivity graphs of (from left to right) linear, ladder, grid, and all-to-all connected devices. Physical qubits are represented by the nodes while the edges correspond to the possible locations of two-qubit gates. The labels indicate the physical qubit indices which will be used in our algorithm presentations.

quadratically faster than both of these.

Fig. 1 shows the scaling of these optimized algorithm designs and compares them to the increase in physical complexity of the underlying machines.

To normalize the effects of native gate set support in different physical architectures, we use a standard gate set comprised of the Hadamard gate and all other single-qubit rotations, and the CNOT gate. All other operations are decomposed into this set, (e.g. SWAP operations decompose into three successive CNOT operations), and these decompositions are described both throughout the text and in Appendix 6.9.

6.3 Hardware architectures

Limited connectivity graphs can impose a large overhead if several SWAP operations are required in order to complete the execution of every gate. To quantify the impact of the connectivity, we consider four typical hardware architectures and describe them as connectivity graphs (here each node represents a physical qubit and each edge indicates the availability of two-qubit gates between the connected qubits). The architectures we chose to analyze were selected because they each have either been physically realized in hardware or are anticipated to be fabricated in the near future. For a total of n qubits, and from the most constrained to the least constrained connectivity, one has:

linear Linear nearest neighbor graph with open boundaries and $n-1$ edges. In solid-state implementations, qubit control lines here could be in the same plane making this easy to manufacture. This has been physically realized with trapped ions [94], superconducting qubits [124, 18] and quantum dots [231].

ladder Grid of dimensions $n/2 \times 2$ with $3n/2-2$ edges. One can visualize it as comprised of two columns of $n/2$ qubits each. Physical realizations include IBM's Quantum Experience [3] as well as Google's "Foxtail" architecture [6].

square grid Square grid of dimensions $\sqrt{n} \times \sqrt{n}$ with $2(n-\sqrt{n})$ edges. An extension of this design is naturally suitable for topological error correction, e.g. by encoding information via the surface code [4, 139]. It likely requires the development of out-of-plane control lines to address the qubits in bulk.

all-to-all Fully connected graph with $n(n-1)/2$ edges. No need for routing. It has been implemented in small trapped ion systems [49], but likely infeasible for large number of qubits. Often used in the abstract description of algorithms.

If the number of qubits is such that a ladder or square grid is not fully filled, one can consider the smallest integer larger than n that would complete the structure in the above estimates. In addition, all of these designs are assumed to provide complete and independent qubit control, meaning that one can perform gates in parallel if and only if they act on different qubits.

As indicated in the description, these architectural design choices are also intended to sweep the physical fabrication complexity spectrum, with the most constrained connectivity graphs being the easiest to fabricate. Graphically, the architectures are shown in Figure 6.2. In this work we analyze the performance of each of these architectures with respect to three important quantum applications and subroutines, described in the next section.

6.4 Results

In this section, we map and schedule the benchmarks discussed above on each of the qubit connectivity graphs under consideration. These algorithms are presented in high level overviews, followed by pseudocode descriptions, followed lastly by analysis.

Two goals are balanced in the selection of the algorithms to study: representation of a large portion of all quantum algorithms, and algorithmic complexity class separation. Specifically, we select quantum subroutines that are present as a significant component of many other quantum algorithms so as to cover a significant portion of the benchmarks. Additionally we seek to ensure that algorithms with both *exponential* and *polynomial* speedup with respect to their classical counterpart algorithms, are represented. This choice allows our results to speak to the impact that particular hardware architectural constraints impose upon quantum speedups at different scales.

The three quantum algorithms that we study are the Quantum Fourier Transform (QFT), Jordan-Wigner Transformation (referred to as the "rotation based on parity"), and the Grover Diffusion Operator comprised primarily of a single multi-control phase operation. The QFT [155] is the quantum analog of the classical discrete Fourier transform, performing a fourier transformation on an input vector of complex numbers. The Jordan-Wigner Transformation is a method by which a system of fermions is mapped onto a system of qubits, respecting (anti-)commutation relations, and is important for many molecular simulations and materials studies. The Grover Diffusion Operator represents algorithms with *polynomial* speedup, and can be described as a routine in which a unique target element is located in an unstructured search space.

We use the convention that any operation contained in a block labeled by **parallel** will be executed in parallel with all of the other gates contained in the *unrolled* version of the block. Blocks may include conditional or loop statements, like "while" and "for" instructions. Unrolling the blocks amounts to resolving all indices of any operation contained in the block, and inlining the resulting gates.

As mentioned previously, to standardize the analysis across these benchmarks, we describe all circuits in terms of a native gate set described in Appendix 7.3 comprised of the Hadamard gate, single-qubit rotations, and the CNOT gate. All of the algorithmic benchmarks are decomposed into this basis, and gate counts and circuit depths are reported in terms of these operations. Observe that the SWAP operation is decomposed into a sequence of three CNOT gates, while the controlled phase and Toffoli gate are decomposed as indicated in the descriptions of the benchmarks in Appendix 6.8 and shown explicitly in Appendix 6.9. These decompositions are utilized in resource overhead calculations, but are omitted in pseudocode algorithm descriptions for clarity.

6.4.1 Linear Array Results

Quantum Fourier Transform

We first show that the Quantum Fourier Transform on a linear array requires $10n - 13$ operation steps. A similar approach was presented in [71] as part of the circuit implementing Shor's algorithm, but with a gate set containing arbitrary 2-qubit gates.

The initial placement of the qubits is the trivial one, with each logical qubit x_i mapped to the physical qubit q_i. We order the qubits from top to bottom as in Fig. 6.2. The implementation proceeds by performing the first Hadamard on the top qubit (index 1), followed by a controlled-phase (denoted by CP in the following) gate between this qubit with its neighbor. This is followed by a SWAP between the top two qubits. The sequence is repeated on the top qubit which now has logical index 2. Meanwhile logical qubit 1 is available for a controlled-phase gate with logical qubit 3. With each repetition of this pattern, the number of SWAP and CP gates performed in parallel is increased by one. The scheme is presented in Algorithm 2, and visualized in Fig. 6.3.

Performance Analysis: In the structure of the algorithm, there is a single outer loop iterating $2n - 3$ times. Each iteration of this loop is comprised of two separate inner loops.

Algorithm 2 Linear Quantum Fourier Transform

logical qubit x_i mapped to physical qubit q_i

```
1: H (q_1)
2: for i in {2, 3, 4, ..., n − 1, n, n − 1, ..., 2} do
3:     j ← i
4:     while j ≥ 2 do Parallel
5:         CP_j (q_j, q_{j−1})
6:         j ← j − 2
7:         if j == 1 then
8:             H (q_1)
9:         end if
10:    end while
11:    j ← i
12:    while j ≥ 2 do Parallel
13:        SWAP (q_j, q_{j−1})
14:        j ← j − 2
15:    end while
16: end for
17: H (q_1)
```

Each of these loops is completely parallelizable, such that the circuit depth of any iteration of the outer loop is equal to the summation of the depth required to execute a single gate from both of the inner loops, which after gate decomposition requires a total of 7 time steps. Notice however that a single CP_k gate followed by a SWAP gate on the same qubit operands offers a circuit optimization in which two inner CNOT operations of the decomposed version of this gate sequence cancel. Because of this, the depth of a single iteration of the outer loop is reduced by 2. Added to this are the leading and final Hadamard operations, resulting in a final running time of $10n - 13$.

Jordan-Wigner String

The Jordan-Wigner transform is analyzed using the proxy benchmark of rotations based on parity operators. For a quantum machine with linear nearest neighbor qubit connectivity, any parity-based rotation can be implemented with a circuit of depth at most $(n + 1)$ or $(n + 2)$, respectively for n even or odd. This includes rotations based on any product of Pauli

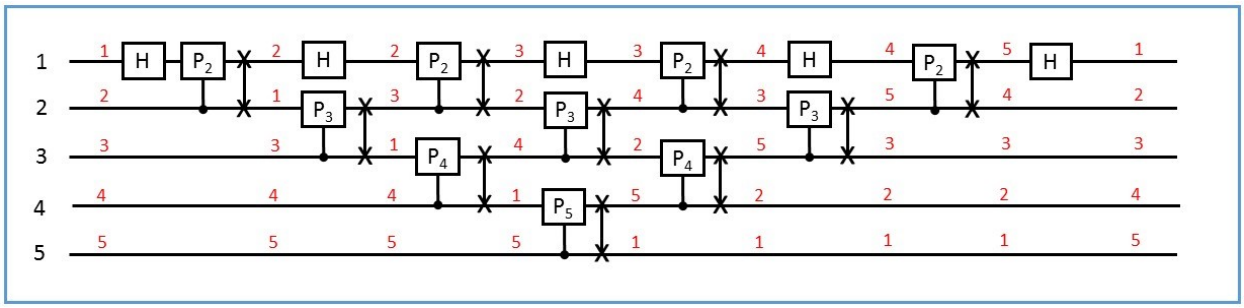


Figure 6.3: The Quantum Fourier Transform on a linear array, as described by Algorithm 2. The physical index of the qubits is noted in black at the left end of the quantum circuit, while the red labels correspond to the logical indices which vary during the circuit due to the extra SWAP operations. Time runs from left to right. Graphically each horizontal line represents a single physical qubit, and each box represents a quantum operation. Vertical lines connecting two physical qubits are two-qubit operations, with "X" connections indicating SWAP operations, and solid circles indicating controlled operations. For more detail see Appendix 7.3.

operators acting on any subset of qubits within the machine.

We consider two cases for the PAR operator, depending on whether it involves all or a subset of the qubits. In the first case the optimal approach corresponds to the CNOT staircase presented in Appendix 6.8, but starting from both the end points of the chain and performing the rotation on the physically central qubit. Pseudo-code for this construction is provided in Algorithm 3, and a visualization is included in Fig. 6.4. Notice that it works irrespective of how the logical qubits are mapped to the physical register.

For future reference we refer to the first half of Algorithm 3 by PARITY, and its inverse operation by PARITY†. This subroutine stores the parity of the involved qubits in the central qubit, and will be used as a component of later algorithmic constructions.

More interesting is the situation where PAR involves a randomized subset of the qubits, possibly spread over the full machine and including physically disconnected qubits. Consider the physical-to-logical index map $f : \mathbb{N} \to \mathbb{N}$ such that $f(i) = j$ indicates that physical qubit q_i is associated to logical qubit x_j. In addition, we introduce the function $p : \mathbb{N} \to \{0, 1\}$ that for each index j returns $p(j) = 1$ if logical qubit x_j is part of PAR or $p(j) = 0$ otherwise.

With these definitions, it is straightforward to modify the pseudocode to the new situation. In practice, substitute each CNOT(q_i, q_j) operation for which $p(f(j)) = 0$ with SWAP(q_i, q_j).

Algorithm 3 Linear Jordan-Wigner String: $e^{-i\frac{\theta}{2}\sigma_z^{\otimes n}}$

arbitrary map between logical and physical qubits

$m \leftarrow \lfloor n/2 \rfloor$
$k \leftarrow n \pmod 2$
$i \leftarrow 1$
while $i < m$ **do Parallel**
    CNOT (q_i, q_{i+1})
    CNOT (q_{n-i+1}, q_{n-i})
    $i \leftarrow i + 1$
end while
CNOT (q_{m+1+k}, q_{m+k})
if $k == 1$ **then**
    CNOT (q_m, q_{m+1})
end if
Rz (q_{m+k}, θ)
if $k == 1$ **then**
    CNOT (q_m, q_{m+1})
end if
CNOT (q_{m+1+k}, q_{m+k})
$i \leftarrow i - 1$
while $i > 0$ **do Parallel**
    CNOT (q_i, q_{i+1})
    CNOT (q_{n-i+1}, q_{n-i})
    $i \leftarrow i - 1$
end while

The circuit can be optimized by eliminating initial SWAPs at the edge of the chain between physical qubits not involved in PAR. Finally, since the duration of CNOT and SWAP may differ (for example, a single SWAP is usually decomposed in terms of 3 consecutive CNOT gates), it is better to execute the gates from the end points in parallel even if in an asynchronous manner.

Performance Analysis: Walking through the algorithm, the general case is executed with complexity $\mathcal{O}(n)$. The worst case performance is dependent upon the relative gate durations of the CNOT and the SWAP operations: if CNOTs require more time overhead than SWAPs, the worst case is achieved when PAR involves all qubits. If instead SWAPs take longer to execute than CNOTs, the worst case occurs when PAR involves only the two end-points of the linear machine respectively. Formally, the latter situation corresponds to

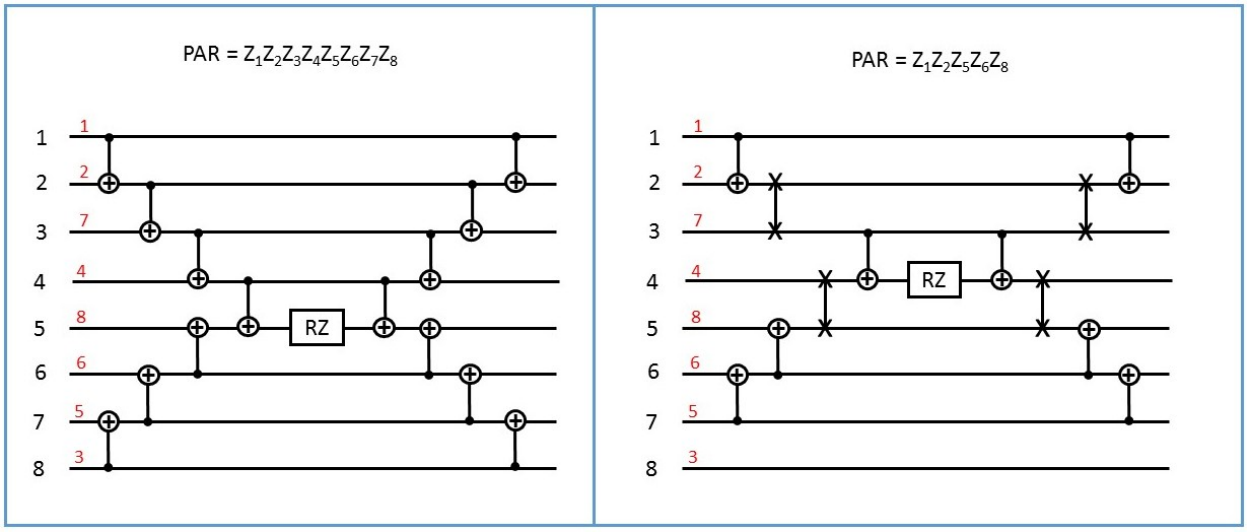


Figure 6.4: Schedule for the parity-based rotations with 8 qubits. The physical index of the qubits is noted in black, while the red label corresponds to the logical index. SWAP gates result in the exchange of logical qubit indices immediately after execution. Shown is an example involving an initial non-trivial placement of logical qubits onto the physical qubit graph. Left panel: circuit that calculates global parity. Right panel: parity only involves logical qubits $\{x_1, x_2, x_5, x_6, x_8\}$.

PAR=$Z_i \otimes Z_j$ with $\{i, j\} = \{f(1), f(n)\}$. The exact cost depends on the distance between the two farthest "active" qubits in the chain and the ratio between the required CNOT and SWAP gates. In cases with required SWAP operations, the running time can be parameterized directly by inlining the gate decomposition and counting these operations as $n+1+(n \bmod 2)$, or $N_{\mathrm{CNOT}} + 3N_{\mathrm{SWAP}} + 1 + (n \bmod 2)$, where N_{CNOT} and N_{SWAP} denote the total number of logical CNOT and SWAP operations in the algorithm, respectively. In the results table 6.1, the depth is reported for an instantiation of the algorithm with global parity.

Grover Diffusion Operator

As mentioned in section 6.8, the main subroutine of Grover's quantum search algorithm computes the logical AND of n bits using a sequence of Toffoli gates that can be executed in circuit depth $\mathcal{O}(\log n)$, each of which has a small constant expansion into one and two qubit gates. To schedule this circuit, consider a *binary tree* containing a total of $(2n-1)$ nodes: the n leaf nodes represent the data qubits, while all other nodes correspond to ancilla qubits

Algorithm 4 PARITY Subroutine

1: $m \leftarrow \lfloor n/2 \rfloor$
2: $k \leftarrow n \pmod 2$
3: $i \leftarrow 1$
4: **while** $i < m$ **do Parallel**
5: CNOT (q_i, q_{i+1})
6: CNOT (q_{n-i+1}, q_{n-i})
7: $i \leftarrow i + 1$
8: **end while**
9: CNOT (q_{m+1+k}, q_{m+k})
10: **if** $k == 1$ **then**
11: CNOT (q_m, q_{m+1})
12: **end if**

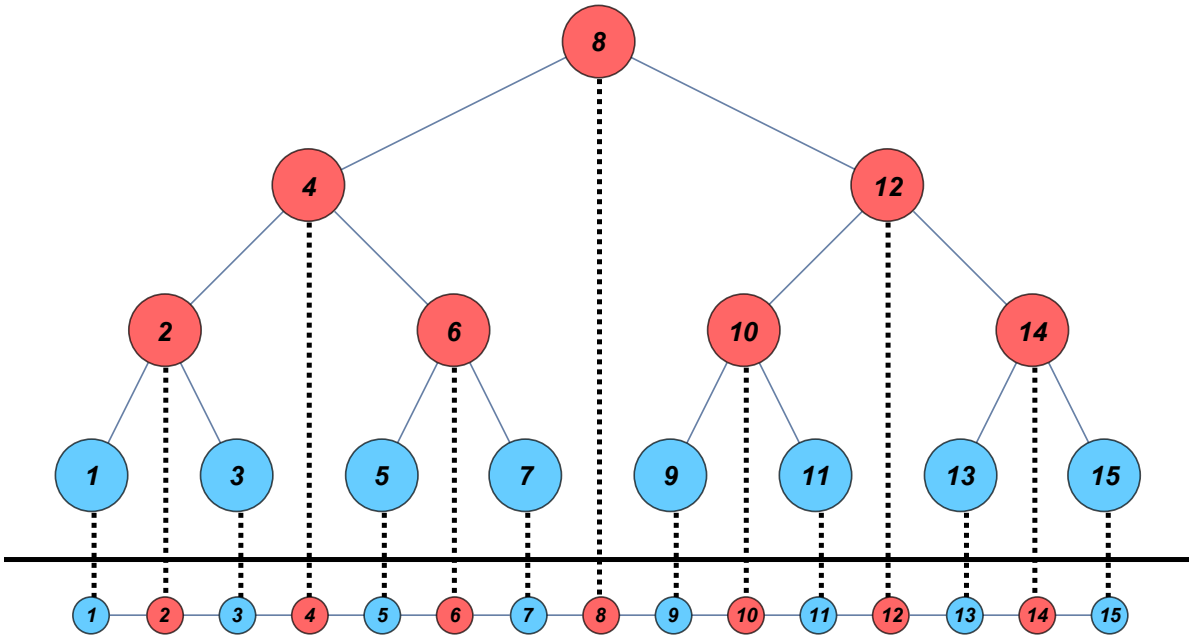


Figure 6.5: Inorder transversal linearization of a binary tree. The color code of the nodes reflects their logical roles: data qubits are colored blue, while ancilla qubits are colored red.

initialized in state $|0\rangle$. The desired Grover diffusion operator can be realized by operating from the leaf nodes up through the tree, by performing a Toffoli gate targeting the root of any subtree and controlled by the two children nodes of the subtree. Each intermediate node contains the result of the logical AND between all leaves of the subtree that has it as the root, and therefore the unique root of the full binary tree contains the logical AND of all the data qubits.

The schedule for a linear connectivity is obtained by linearizing the binary tree with in-order transversal indexing, as visualized in Fig. 6.5. We consider the case $n = 2^k$ for convenience, but the algorithm can be easily adapted beyond this condition by considering $k = \lceil \log_2(n) \rceil$. Data qubit x_i, with $i = 1, 2, \ldots, n$, is mapped to physical qubit q_{2i-1}. The ancilla qubits forming the level immediately above in the binary tree (i.e. having height 1) correspond to physical qubits q_{4j-2} for $j = 1, 2, \ldots, n/2$. Ancilla qubits at greater heights are similarly uniformly distributed in the physical qubit chain.

The Toffoli gates at the lowest height of the binary tree can be immediately implemented on the linear configuration, as each pair of control data qubits is adjacent to the target ancilla between them. However, one needs to apply SWAP gates to route the ancilla qubits together to connect the physical qubits that compose the Toffoli gates for the higher levels of the binary tree. This is done by keeping the target qubit of the Toffoli gate fixed, and by moving the two control qubits (one on each side of the target qubit) to the adjacent positions in the linear array of qubits. The pseudocode is presented in Algorithm 5 and graphically in Fig. 6.6.

Performance Analysis: With an in-order traversal mapping of the binary tree, we can analyze the SWAP overhead between each round of Toffoli gates, and calculate a final performance overhead. In the level at height h of a $k = \log_2 n$ depth binary tree, where h runs from $1, \cdots, k$, the distance between the ancilla comprising each Toffoli triplet requires $(2^h - 2)$ SWAPs to be covered[3], given by the size of the subtrees that are located between these nodes in an inorder traversal. These can be executed in parallel, as we can SWAP the control qubits both towards the target qubit, which is located between the controls by design. This results in a SWAP chain circuit depth of $2^{\log(n)-h} - 1$ for the iteration at height h. For an entire binary parity tree then, we add to the parity accumulation circuit a total circuit depth of $\sum_{h=1}^{\log(n)} 2^{\log(n)-h} - 1 = n - \log(n) - 1$. Adding these to the $\log(n)$ serial Toffoli gate blocks,

3. The depth of a node is the number of edges from the node to the tree's root node. A root node will have a depth of 0. The height of a node is the number of edges on the longest path from the node to a leaf. A leaf node will have a height of 0.

again each decomposed into a set of 4 single qubit rotations and 3 CNOT operations, results in a full algorithm running time of: $2(3(n-1-\log_2(n))+7\log_2(n))+1 = 6n+8\log_2(n)-5$ time steps, including the execution of the inverted gate sequence and the single qubit gate in the center of the circuit. Crucially, we find that this does not impede the quadratic speedup offered by the algorithm over the classical search counterpart algorithm, given sufficient size of the input space. In fact the overhead is $\mathcal{O}(n)$ while the quantum speedup was $\mathcal{O}(2^{n/2})$.

Two final remarks: first we observe that the initial mapping between logical data qubits and the physical qubits can be relaxed. Since the Grover diffusion operator is a symmetric operation of all data qubits, any initial mapping that alternates a data qubit with an ancilla qubit works equally well. Second, we notice that such configuration can be obtained with at most $n(n-1)/2$ SWAPs and depth of $(n-1)$ sequential SWAPs, the worst initial mapping being all data qubits accumulated at one end of the linear array.

6.4.2 Ladder Results

Quantum Fourier Transform

For the QFT, a simple embedding of the linear array onto the ladder of two columns each of height $\frac{n}{2}$ would allow for the application of Algorithm 2 directly, which would not reduce circuit depth. We improve upon this by designing a schedule that offers a constant-factor speedup over a linear array. We label the physical qubits as shown in Fig. 6.2. This labeling scheme will be used to enable direct logical to physical qubit mapping. For this, we will essentially execute two copies of Algorithm 2 in parallel, one on each column of the ladder, interspersed with controlled-phase gates between neighboring qubits across the ladder. The procedure is presented in Algorithm 6 and Fig. 6.7.

Performance Analysis: This algorithm is composed of $n-1$ iterations of 3 types of parallel blocks. First, we execute a set of "horizontal" controlled phase gates that cross the ladder, followed by a set of "vertical" controlled phase gates. Lastly we immediately follow

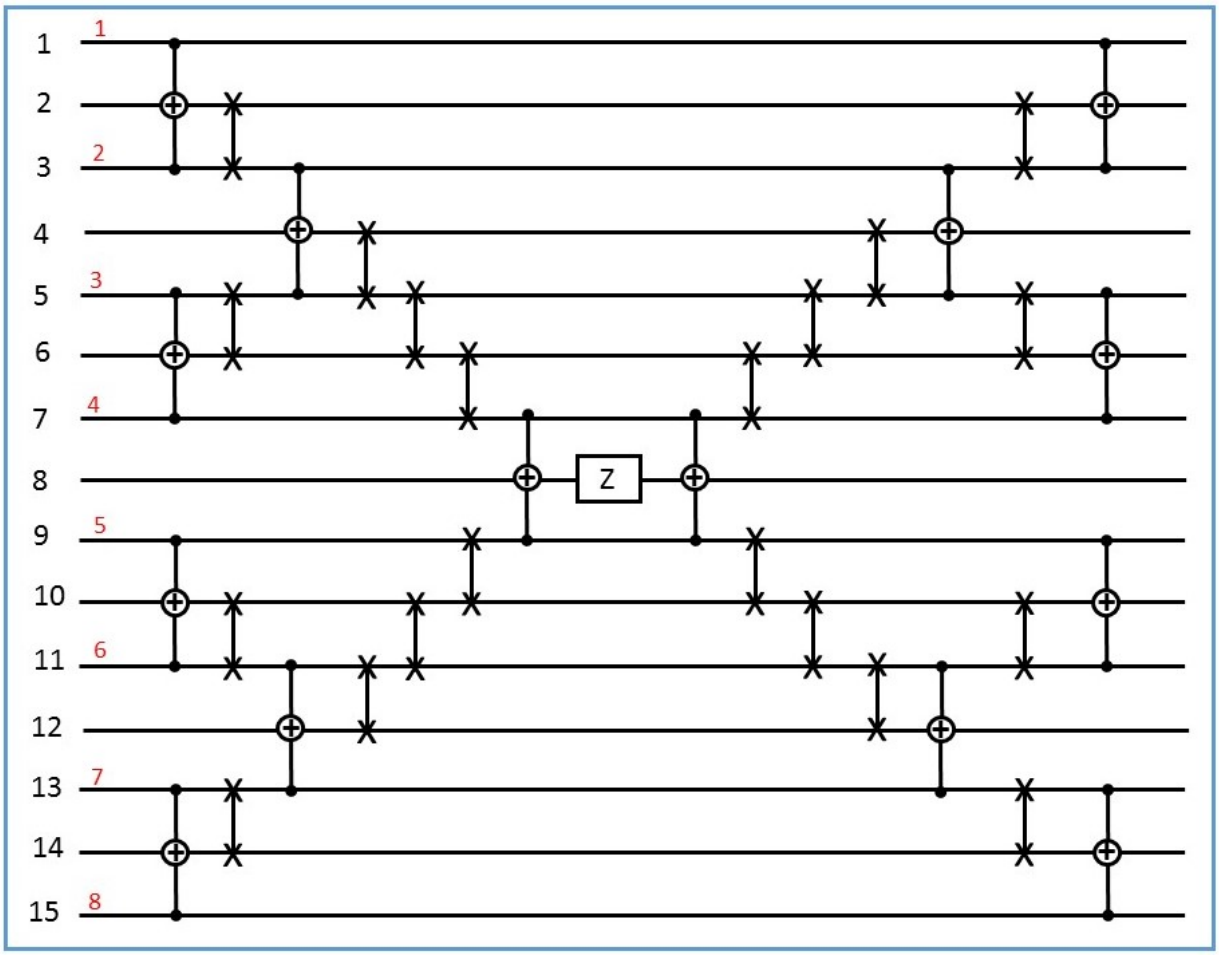


Figure 6.6: Schedule for the Grover diffusion operator with $n = 8$ data qubits, corresponding to a database with $2^8 = 256$ elements. The implementation also includes $(n - 1) = 7$ ancilla qubits, those not associated with a red index.

the vertical gates with a corresponding set of SWAP operations. The SWAP overhead has been reduced, with respect to the linear case, by only inserting a SWAP timestep once for every two sets of controlled phase operations. The first $n - 2$ iterations of the algorithm execute all 3 of these blocks, for an iteration depth of 9 after canceling the internal CNOTs as was performed for Algorithm 2. The last iteration only executes the first of these blocks, adding 4 operations to the critical path, which combines with a final step of SWAP operations and with leading and final Hadamard operations for a total circuit depth of $9n - 11$, a small reduction compared to the linear nearest neighbor machine.

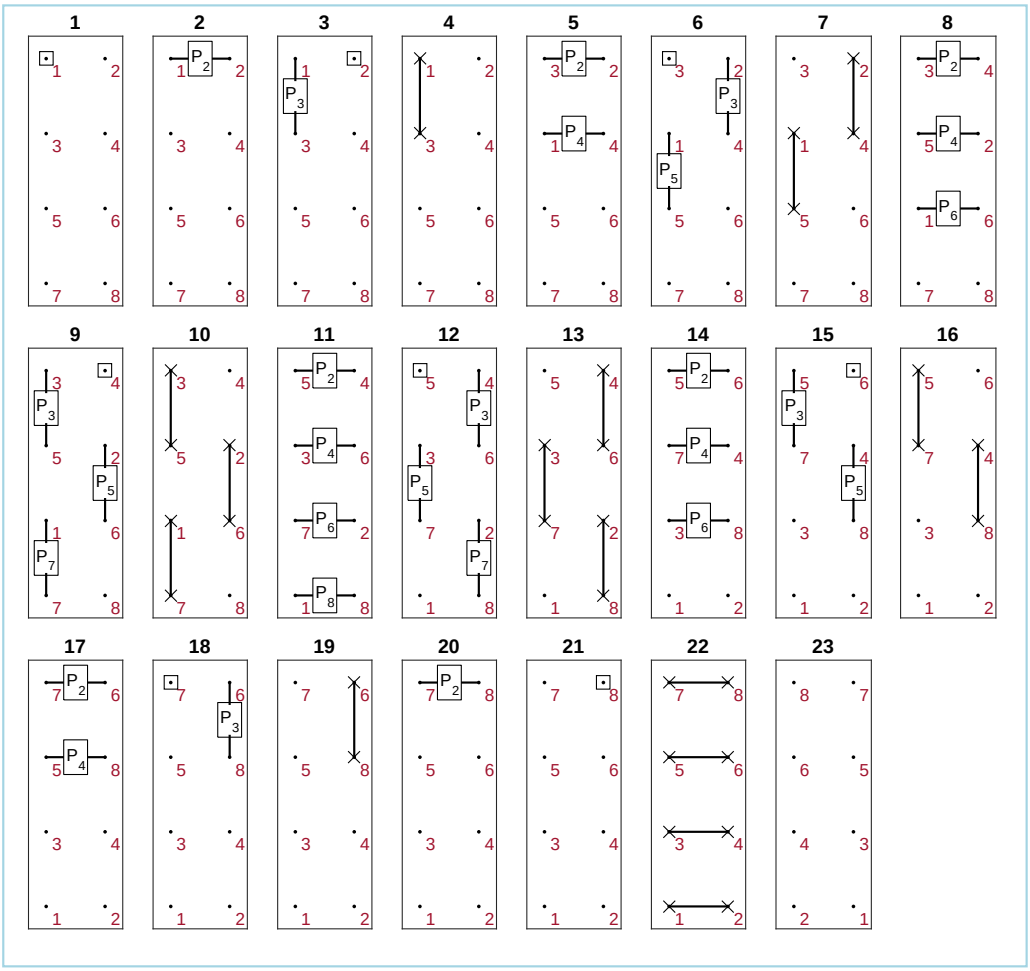


Figure 6.7: Snapshots in time of the QFT being executed on a ladder. The squares represent Hadamard gates. The two-qubit gates used are controlled-phase and SWAP gates. The numbers on top of each sub-figure indicate the temporal order.

Jordan-Wigner String

A Jordan-Wigner string on a ladder can be executed in time asymptotically equivalent to that of the linear array with a factor of 2 reduction. The construction proceeds by implementing the PARITY subroutine defined in Algorithm 4 in parallel on each of the two columns of height $n/2$ of the ladder, adding an additional CNOT operation between the center two qubits located in row $\lfloor n/4 \rfloor$. The rotation operation is performed on the target of this added CNOT, and the circuit is then reversed. This is described by the pseudocode contained in

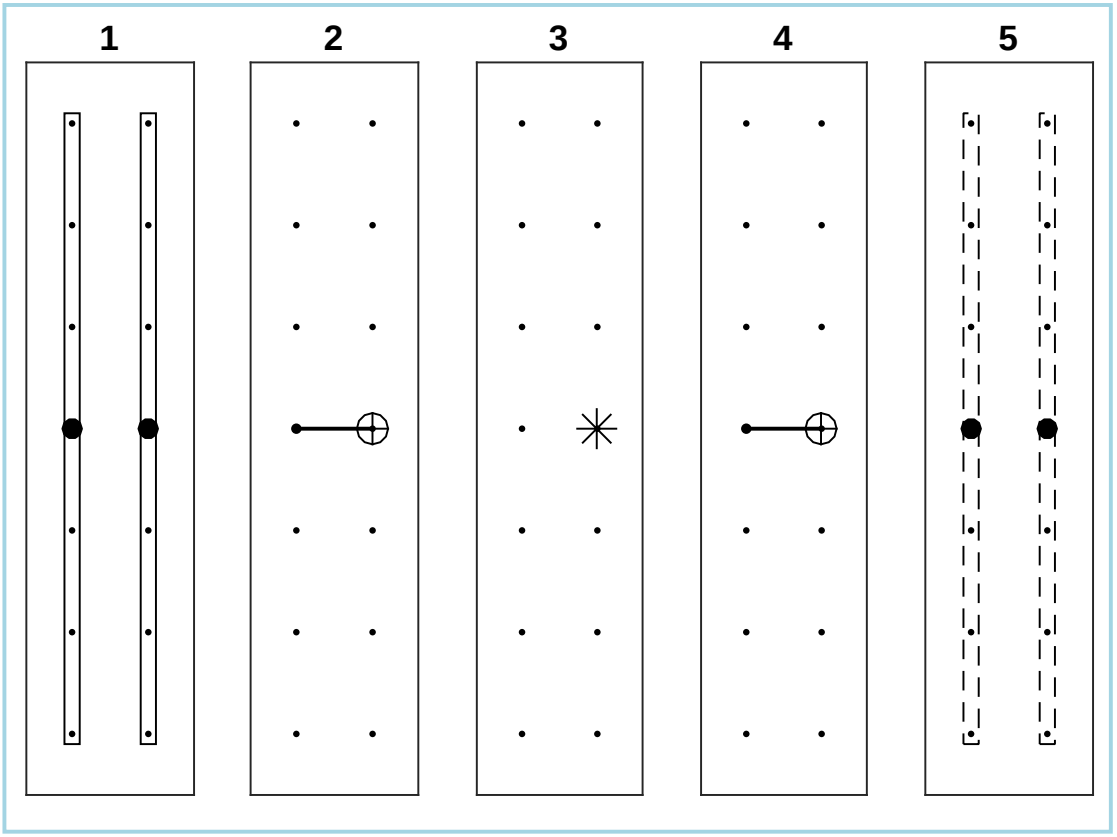


Figure 6.8: Schedule for the Jordan-Wigner string on a ladder. The solid vertical box is the PARITY operation on a linear array and the dashed vertical box is PARITY†. There are CNOT gates between the middle qubits on either leg of the ladder and the $*$ operator indicates a Z-rotation.

Algorithm 9, presented in Appendix 6.9.3.

Performance Analysis: By splitting the connectivity into two columns each of height $n/2$, an approximate factor of 2 in circuit depth can be saved by executing Algorithm 3 on each column in parallel, resulting in a final running time of $n/2 + 1 + (n/2 \mod 2)$, where as before n can be parameterized by N_{CNOT} and $3N_{\text{SWAP}}$ in order to account for the SWAP gate decomposition. Reported in table 6.1 is the circuit depth of an instantiation of the algorithm with global parity.

Grover Diffusion Operator

To utilize the added connectivity of the ladder, we will consider an extension of the binary parity tree in-order traversal embedding given in section 5. We find that the added connectivity here also allows for approximately a factor of 2 reduction in the total SWAP count and the circuit depth of the resulting circuit. Intuitively, the idea is to use a ladder of two columns each of height n and to extend the linear embedding by first embedding all of the n data qubits into the first column, and using the second column as ancilla. The first operation of the algorithm is to perform the first Toffoli interaction between every pair of input qubits, targeting a single ancilla qubit in the second column. The resulting ancilla qubits in the second column are then spaced out by another yet unused ancilla qubit. This operation requires an additional 2 SWAPs added to the circuit depth of the Toffoli gate in decomposition, to account for the non-adjacency of the one of the control qubits and the target qubit. The remainder of the algorithm implements the linear array algorithm on the second column, which amounts to executing a new binary parity tree subroutine of depth $k' = \log(n/2) = k - 1$.

Pseudocode for this construction is specified in Algorithm 7, noting that the input qubits are positioned in the first column, and the ancilla are all positioned in the second column. The physical indices will be specified again as in Fig. 6.2.

Performance Analysis: The algorithm operates by performing the lowest level of the binary parity tree interaction first, and embedding these results into a linear array fully contained in the second column of the ladder. The remainder of the algorithm is to complete the binary parity tree, which can be completed by directly performing Algorithm 5 on this second column.

Notice that, by doing this, the second lowest level of the parity tree becomes adjacent as well. In fact, the performance of this algorithm is equal to the performance of Algorithm 5 operating on a set of $n/2$ input qubits, with an additional two Toffoli operation steps. The ladder can perform the full algorithm in circuit depth $6(n/2) + 8\log_2(n/2) - 5 + 14 + 12 =$

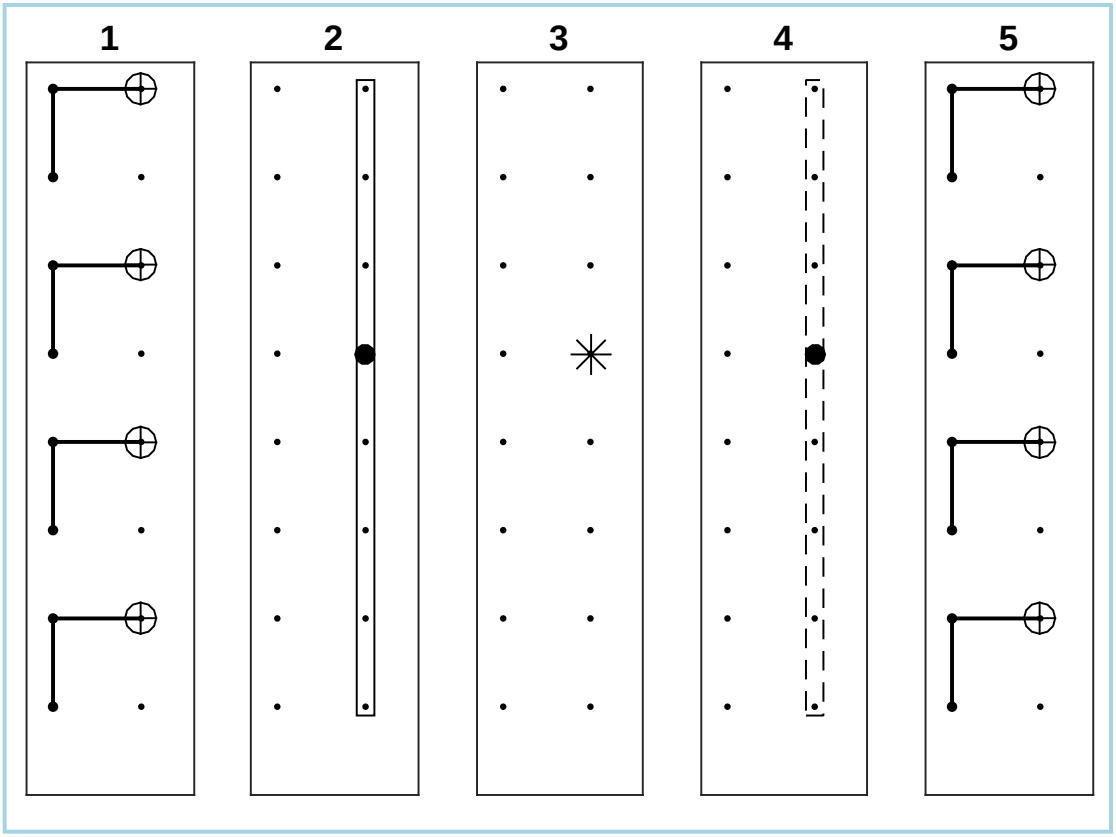


Figure 6.9: Schedule for the Grover diffusion operation on a ladder. Toffoli gates of steps 1 and 5 are explicitly depicted. Step 3 includes a single Z. The solid box is the AND operation on a line and the dashed box is AND†.

$3n + 8\log_2(n) + 13$, reducing the circuit depth compared to the linear array by approximately a factor of 2.

6.4.3 Grid Results

Quantum Fourier Transform

A trivial embedding of the linear algorithm 2 enables the grid topology to achieve $10n - 13$ circuit depth. Any embedding that preserves adjacency of sequential logical indices will be equivalent in performance up to SWAP steps required recover the initial mapping. An example of a trivial linear embedding maps the logical qubit indices q_i to the grid locations

(row,col) in row major order, mapping logical indices sequentially across each row from left to right, and mapping from right to left in each alternating row.

While an implementation of the square grid QFT algorithm can potentially obtain a lower circuit depth than that of the ladder implementation, all our efforts failed to provide a more compact schedule. Two such efforts are the extension of the linear algorithm by "snaking" through the grid at every third row. While this preserves the ability for each qubit in the line to interact with three neighbors before progressing, it still requires that all qubits traverse the path entirely. This imposes an overhead that exceeds that of the original line algorithm. A second effort was the embedding of the ladder algorithm into the grid. This construction performs a ladder subroutine on the first two columns (rows) of a grid, and then performs a small line subroutine to connect this ladder to the adjacent two columns (rows). In this way the grid simulates a single large ladder.

While this preserves the number of executions of the outer loop in Algorithm 6, we have inserted additional gates to perform the line subroutine that cannot be completely parallelized. As a result the overhead from this construction ultimately exceeds that of the original ladder algorithm, and even the line algorithm, by constant factor. For this reason we report the line embedding overhead in the results Tables 6.1 and 6.2.

Additionally, the performance gap between the linear algorithm (Algorithm 2) and the theoretical lower bound is not likely to justify the complexity that would be required to build the machine. The magnitude of the performance gap between the ladder and the theoretical lower bound is one of the major results of this work: namely that a limited connectivity device (1D or almost 1D) can perform QFT with near-optimal application latency.

Jordan-Wigner String

The 2D-grid can be used effectively to reduce the circuit depth overhead for the Jordan-Wigner string from $\mathcal{O}(n)$ to $\mathcal{O}(\sqrt{n})$ by extending the construction utilized to perform the ladder implementation.

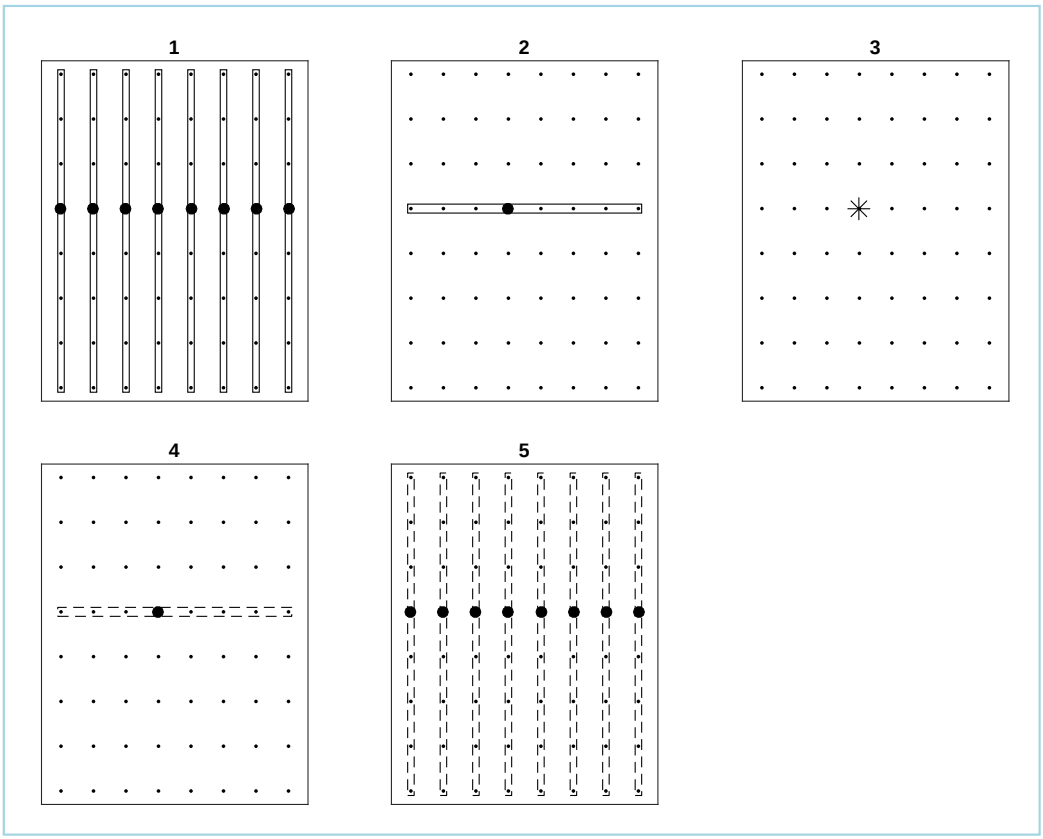


Figure 6.10: Schedule for the Jordan-Wigner string on a grid. The solid boxes are the PARITY operations on a line and the dashed boxes are PARITY†.

Specifically, considering a square grid of dimension $\sqrt{n} \times \sqrt{n}$, the transform can proceed by first performing the PARITY subroutine in Algorithm 4 on all columns (rows) of the grid, followed by an invocation of the Linear Array Algorithm 3 operating on the central row (column).

Performance Analysis: Altogether, this Algorithm is comprised of four serialized sets of parallel PARITY subroutines, each executed on $\sqrt{n}$ input qubits, as well as the central rotation gate. This combines for a full execution time of $2\sqrt{n} + 1 + 2(\sqrt{n} \mod 2))$, with the appropriate adjustments for the presence of SWAP operations.

Grover Diffusion Operation

In order to fully utilize a 2D-grid connectivity for the Grover diffusion operation, we will use both the linear and ladder array implementations as building blocks. Informally, the algorithm will operate on a grid of consisting of $\sqrt{n}$ columns each of length $2\sqrt{n}$, where every column contains $\sqrt{n}$ input qubits and the original $\sqrt{n}-1$ ancilla qubits, and we have inserted an additional row of ancilla qubits into the center of the grid. Every column will perform the AND subroutine. An additional SWAP is inserted to accommodate the additional row. The results of the AND subroutines will be contained in qubits in row $\lceil\sqrt{n}\rceil$, at which point Algorithm 7 is executed on the row containing these results and the added ancilla row. This is summarized by the pseudocode of Algorithm 11 contained in Appendix 6.9.3.

Reference [185] shows that the optimal depth for 2D architectures and non-adaptive circuits is $\mathbb{O}(\sqrt{n})$ and suggest a constructive procedure in which information is processed from the outer square towards the center of the grid. In our case we obtain the same scaling by using our ladder construction as building block.

Performance Analysis: The algorithm can execute all of the column operations in parallel, in time $3\sqrt{n}+4\log_2(n)$ given by the circuit depth of the AND subroutine operating on $\sqrt{n}$ input qubits. It will then perform the subsequent Ladder algorithm in another $3\sqrt{n}+8\log_2(\sqrt{n})+13$ time steps. This is followed by another iteration of a set of parallel column AND subroutines, resulting in a final running time of $9\sqrt{n}+8\log_2(n)+13$. This Algorithm makes good use of nearly the entire 2D grid of qubits that are available, and while there are implementations of these multi-controlled NOT operations that do not require ancilla [17], they add a large expansion to the circuit depth in order to do so. It is likely that an implementation based without added ancilla could save a constant factor of qubits at the expense of a larger expansion in running time. The presented implementation would likely be minimal in comparison, in terms of the product of the number of qubits and the circuit depth required to execute the algorithm.

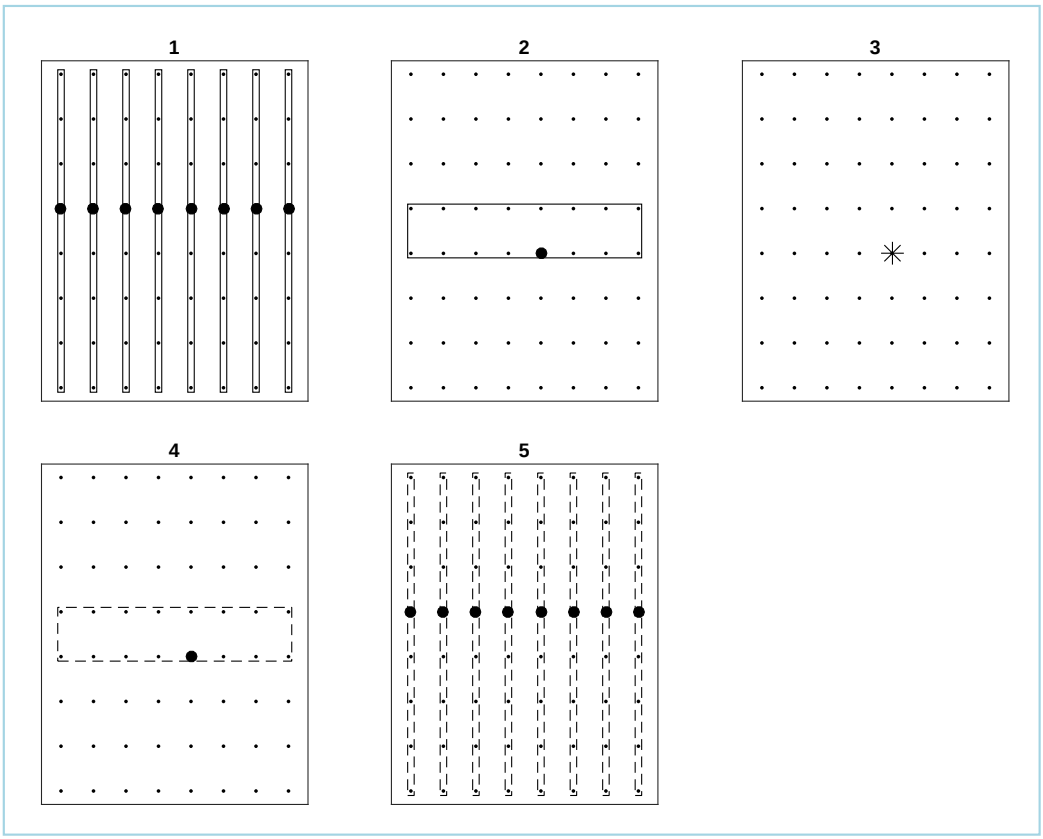
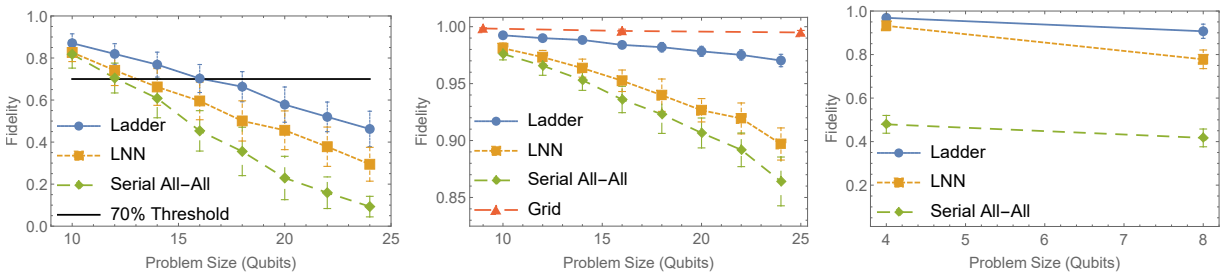


Figure 6.11: Schedule for the Grover diffusion operation on a grid. The solid boxes are the AND operations on a line and the dashed boxes are AND†.

6.5 Performance Evaluation: Realistic Simulations

This section analyzes the performance of these algorithms in realistic simulations. Each algorithm has been implemented in the Intel Quantum Simulator [198] with injected noise to simulate the effects of decoherence in real quantum systems. The goal with these simulations is to evaluate the realistic impact of noisy quantum hardware available today on our schedules. To this end, the simulations are based upon physical error rates reported in recent experimental data [11].

In this study we consider an environmental noise model defined by heuristics T_1 and T_2, representing the relaxation time and dephasing time of each underlying physical qubit



(a) Quantum Fourier Transform (b) Jordan-Wigner String (c) Grover's Diffusion Operator

Figure 6.12: Noisy simulation results for each of the three representative benchmarks considered in this work. Confidence intervals are defined by the standard deviation of the fidelity as calculated by a sequence of Monte Carlo simulations. *Problem Size* denotes the number of data qubits, which is equal to the total system size for both the Quantum Fourier Transform and the Jordan-Wigner Transform. Grover's Diffusion Operator uses $n-1$ additional ancillary qubits. For reference, a serialized version of the algorithms without SWAP operations is provided. This provides a baseline for algorithm fidelity without any added parallelism.

[21, 188]. Under this model, qubits undergo relaxation about the Z axis (e.g. from $|1\rangle \rightarrow |0\rangle$) in time T_1, while phase coherence is lost on the time scale T_2. In our simulations we choose T_1 and T_2 to model recent experimental data [11], specifically simulating a 10^{-3} circuit error rate. This means that there is a factor of 1000 between gate durations and coherence times, implying that 1000 gates can be performed within the coherence time T_1. In other words: $T_1/\Delta t = 10^3$. We set T_1, T_2, and Δt to enforce this property.

To simulate the effects of noise, the circuits are replaced by noisy counterparts in which each gate (or parallel gate set) is followed by the application of a *noise* gate on every qubit. The noise gates sample stochastically from depolarizing error channels. These error channels consist of the application of single-qubit rotations around the X, Y, Z axis after each gate. The angle of each rotation is stochastic and small, drawn from normal distributions with 0 mean and standard deviations given by:

$$s_\alpha = \sqrt{-\ln(1-p_\alpha)}$$

where $\alpha \in \{x, y, z\}$ and

$$p_x = p_y = \frac{1 - e^{\frac{-1}{M_1}}}{4}$$
$$p_z = \frac{1 - e^{\frac{-1}{M_2}}}{2} - \frac{1 - e^{\frac{-1}{M_1}}}{4}$$

where $M_\beta = T_\beta / \Delta t$, Δt is the time over which the noise acts between gates, and $\beta \in \{1, 2\}$. Each noise gate in the circuit samples from this channel, and the corresponding synthesized gate is created and applied to the qubit operands in the gate. The entire simulation is thus comprised of a Monte Carlo sequence of repeated random circuits, where noise gates are sampled independently in each iteration. For more details on this method of noise simulation, see [21, 188].

Overall, we find support in simulation for the claim that designing algorithms around hardware constraints enables the expansion of the computational power of today's noisy small quantum machines, and achieves higher fidelity operations. We also find evidence that increasing connectivity from linear nearest neighbor machines to ladder machines provides a significant computational boost, and while 2-dimensional grid based topologies do generally outperform ladders, in many cases ladder connectivities perform nearly as well as and show scaling properties of fidelity similar to their grid counterparts. This encourages hardware architecture designs to target the ladder as a potential next phase for hardware connectivity.

6.5.1 Quantum Fourier Transform

Figure 6.12a shows the results of the Quantum Fourier Transform simulations. Plotted are the described algorithms from the ladder (Algorithm 6), the LNN (Algorithm 2), and a serialized version of the algorithm assuming all-to-all connectivity. Also plotted is a 70% threshold. From these simulations, we find that the ladder algorithm is able to perform QFT on a system of up to 16 physical qubits before dipping below the 70% fidelity threshold. Contrast this with the LNN algorithm, which dips below this line around 12 physical qubits,

we find that the ladder topology is able to effectively boost the computational power of the machine by nearly 4 physical qubits.

6.5.2 Jordan Wigner Transform

Figure 6.12b shows the results of the Jordan-Wigner String algorithmic simulations. The grid (Algorithm 10) and ladder (Algorithm 9) results are in sharp contrast to the LNN (Algorithm 3) and serialized all-to-all algorithms. This scaling highlights the circuit depth complexities reported in Table 6.1, which have meaningful impact on actual hardware and physical realizability of algorithms.

In these simulations, we find that moving from the LNN to the ladder algorithm enables the transformation to maintain above 96% fidelity throughout system sizes up through 24 physical qubits, while the LNN algorithm decays below 90% at the upper end of these systems. The scaling of this fidelity decay is important as well, and implies after linear extrapolation that the grid, ladder, LNN, and serialized algorithms dip below the 70% threshold at system sizes of 172, 96, 26, and 20 qubits, respectively.

6.5.3 Grover Diffusion Operation

For the Grover Diffusion Operation shown in Figure 6.12c, there are a limited number of valid configurations that can be run in a tractable simulation. This is because, for an input of k qubits, the algorithm as written requires that k be a multiple of 2^m for some integer m, and the total qubit count including ancilla is $2k - 1$. Simulations are able to be completed for the ladder, LNN, and serialized algorithms at sizes up to 16 total qubits, and we see once more that the scaling coefficients derived in Table 6.1 directly correspond to increases in fidelity when the simulations are performed on realistic, noisy hardware.

6.6 Conclusion and Discussion

This work addresses the important question of quantifying the impact of reduced connectivity of hardware architectures on the performance of quantum algorithms. While many quantum applications have been theoretically proposed without concern for the underlying hardware connectivity, we have shown that for three representative benchmarks, extensive optimization is possible when scheduling to devices of limited connectivity. For the Quantum Fourier Transform in particular, the performance gap between a ladder connectivity graph and that of the all-to-all theoretical lower bound is particularly small, corresponding to an overhead in the constant factor of only around 12.5%. However, the other benchmarks we considered show $\mathcal{O}(\sqrt{n})$ performance gaps between ladder and full 2-dimensional grid connectivities, typically related to the maximum distance between any two qubits in the hardware. The primary driver behind these performance figures is scheduling, and as a result it seems that effort spent compiling applications to machines can help reduce the burden on efforts to increase the connectivity of the devices themselves. Specifically, we find in simulation that ladder connectivity graphs often perform nearly as well as grid-based hardware connectivity. This finding would encourage hardware designs to target these ladder topologies, as they can potentially produce the greatest expansion of computation for a modest increase in fabrication complexity.

We find that quantum speedup is well-preserved across each of the algorithms considered even on the most limited qubit connectivity graphs. Specifically, the circuit depth for the Quantum Fourier Transform has only a constant-factor overhead on a limited connectivity graph compared to an all-to-all connectivity graph. The circuit depths for the Grover diffusion operator and the Jordan-Wigner transform are polynomial in the number of qubits n and thus logarithmic in the problem size $N = 2^n$.

While the bounds on the performance impact are relatively small, an increase of a factor of $\sqrt{n}$ may potentially become an impediment to operation of large scale applications in near term devices where the total coherence time available is finite. However, under the

safe assumption that large machines will ultimately need to be operated in a fault-tolerant manner, the extra $\sqrt{n}$ time complexity would only impact the actual execution time of applications, and does not factor into application fidelity. It is reasonable to assume that users of such machines will not be significantly affected by an extension of running time of this magnitude, which is at most linear in the number of qubits and logarithmic when compared to both the quantum speedup and the computational complexity of the corresponding classical algorithms.

Our work extends to the scheduling and execution of fully error corrected circuits operating on encoded qubits. In larger machines, while the underlying physical qubit connectivity may be required to be quite dense to support error correcting codes [53, 28], many proposals for large scale architectures rely upon some form of clustering or modularity of logical qubits [39, 117]. These scheduling results can motivate these approaches, showing that sparse connectivity between encoded qubit modules may suffice for attaining quantum speedup.

Future work in this direction will involve taking into account control constraints, for which, in addition to limited connectivity, the parallel execution of quantum gates is subjected to additional restrictions due, for example, to the electronics.

6.7 Background and Notation

6.8 Representative Algorithmic Workloads

6.8.1 Quantum Fourier Transform

The Quantum Fourier Transform (QFT) [155] is the quantum analog of the classical discrete Fourier transform, in which an input vector of complex numbers $\boldsymbol{x} = (x_0, x_1, ..., x_{N-1})$ is transformed to a vector of complex numbers $\boldsymbol{y} = (y_0, y_1, ..., y_{N-1})$ according to:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k/N} \, x_j \tag{6.1}$$

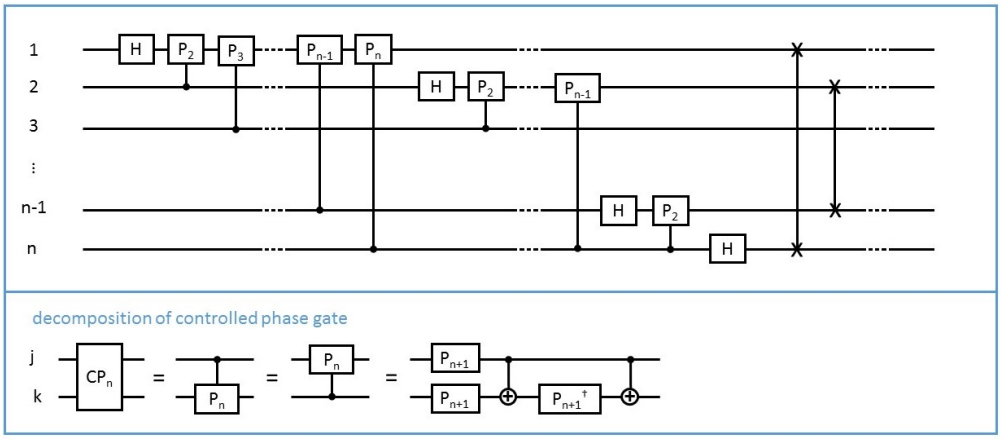


Figure 6.13: Circuit implementation of the Quantum Fourier Transform from reference [155]. The SWAP gates at the end of the circuit are used to invert the qubit order and restore the logical one from Eq. (6.2). The explicit decomposition of the controlled phase gate into single-qubit phase gates and CNOT gates is included in the bottom panel.

The Quantum Fourier Transform takes an input state from the computational basis, namely $\{|0\rangle, \ldots, |N-1\rangle\}$, and transforms it according to:

$$|k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k/N} \; |j\rangle \tag{6.2}$$

For $N = 2^n$, the circuit representation of the QFT requires n qubits and is provided in Figure 6.13 following the usual description [155]. Note that the most significant bit corresponds to the upper most qubit in Figure 6.13: $(n-1)$ controlled phase gates act on logical qubit 1, $(n-2)$ on logical qubit 2 and, in general, $(n-k)$ on logical qubit k, combining to a total $\mathcal{O}(n^2)$ gates to perform the transformation on 2^n numbers. While the number of gates is $\mathcal{O}(n^2)$, the depth of the circuit on an all-to-all connectivity graph can be shown to be $\mathcal{O}(n)$ which sets the theoretical lower bound of any schedule of the operations of the QFT.

As a quantum subroutine, the QFT is an important component of many quantum algorithms including Shor's prime factorization and discrete logarithm algorithms [197], the estimation of eigenvalues and eigenvectors of matrices [7], and others. In particular, the QFT can be seen as a general routine needed to solve problems that can be reduced to the *hidden subgroup problem*, of which the listed examples above are special cases.

The speedup from the QFT comes from the comparison with analogous classical algorithms for performing the Fast Fourier Transform of $N = 2^n$ numbers. The best known algorithm to perform this computation requires roughly $\mathcal{O}(N \log N) = \mathcal{O}(n2^n)$ operations, while invoking the quantum Fourier transform on the same input only requires approximately $\mathcal{O}(\log^2 N) = \mathcal{O}(n^2)$ operations, resulting in an *exponential* speedup.

6.8.2 Jordan-Wigner String: Rotations based on parity operators

The study of materials and molecules is intimately related to the solution of the associated electronic problem: simulating fermionic systems is exponentially expensive on classical computers, but can be efficiently performed with quantum computers [157, 225]. In fact, quantum chemistry is often regarded as the *killer application* of near term quantum devices [30].

The Jordan-Wigner transform is a method by which a system of fermions, most notably electrons, can be mapped onto a system of qubits. In the mapping, the number of electronic "orbitals" is fixed to be a constant. Due to the Pauli exclusion principle, at most one electron can occupy an orbital. One can assign a qubit to each orbital and associate state $|0\rangle$ to the absence of the electron and state $|1\rangle$ to the occupied orbital. However, fermionic states must satisfy anti-symmetry constraints known as the Fermi-Dirac statistics. From a practical perspective, if one wants to create an electron in a certain orbital, a minus sign may be required depending on the occupancy of the other orbitals: to add an electron in the k-th orbital, the minus sign is necessary if an odd number of electrons are present in the orbitals with index $j < k$.

The most expensive step in fermionic simulations arises from the parity operation needed to implement Fermi-Dirac statistics:

$$\text{PAR } |j\rangle \otimes \cdots \otimes |k\rangle = (-1)^{j \oplus \cdots \oplus k} |j\rangle \otimes \cdots \otimes |k\rangle \tag{6.3}$$

that introduces a minus sign depending on whether the number of involved qubits that are in state $|1\rangle$ is odd or even.

For a single qubit PAR=Z, while in full generality it may involve all qubits. We call "rotation based on parity", or a "Jordan-Wigner string", the operation described by:

$$\exp\left(-i\tfrac{\theta}{2}\mathrm{PAR}\right)\,|j\rangle\ldots|k\rangle = \\ = \begin{cases} e^{-i\pi\frac{\theta}{2}}\,|j\rangle\ldots|k\rangle & \text{if } j\oplus\cdots\oplus k=0 \\ e^{+i\pi\frac{\theta}{2}}\,|j\rangle\ldots|k\rangle & \text{if } j\oplus\cdots\oplus k=1 \end{cases}, \tag{6.4}$$

where θ is an arbitrary angle. In the following, rotations based on parity are considered the appropriate proxy operation to implement fermionic simulations[4]. The decomposition of the rotation into one- and two-qubit gates typically takes the form of a "CNOT staircase" illustrated in Figure 6.14, where $\mathrm{RZ}(\theta) = e^{-i\frac{\theta}{2}Z}$.

Observe that the local basis of each qubit can be changed before and after the parity-based rotation, in this way the circuit implements rotations based on arbitrary products of Pauli operators. The overhead due to a change in basis in circuit depth is minimal and $\mathcal{O}(1)$ irrespective of the number of qubits n.

Finally we observe that, while the CNOT staircase is intuitively simple, without limited connectivity one can dramatically reduce the circuit depth from $\mathcal{O}(n)$ to $\mathcal{O}(\log_2 n)$ by parallelizing the CNOTs according to a tree structure.

6.8.3 Grover Diffusion Operator

The previous two quantum routines are part of quantum algorithms providing exponential speedup over their classical equivalents. To represent quantum algorithms with *polynomial* speedup relative to their classical counterpart, we consider the task of searching for a target

4. The reader familiar with quantum simulation will realize that each factor of the Trotter-Suzuki decomposition of the propagator can be written as a rotation based on parity when suitable single-qubit operators are added to change the local basis.

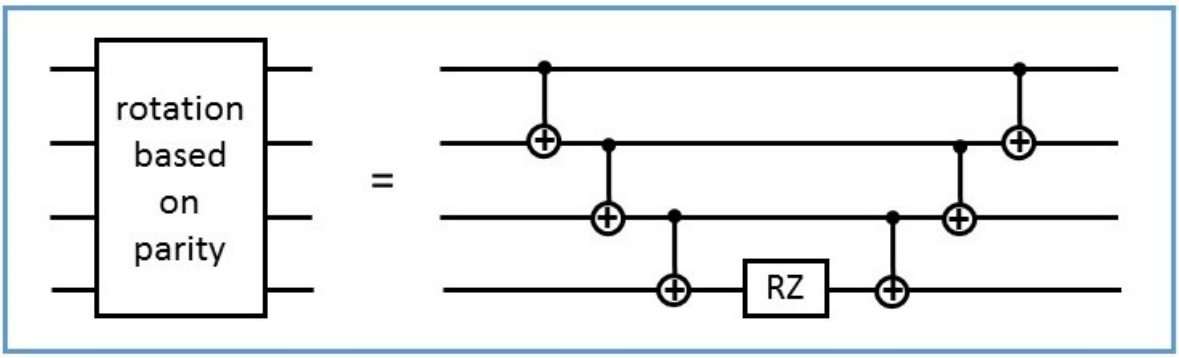


Figure 6.14: Quantum circuit implementing a typical operator used in quantum simulations of fermionic systems via the Jordan-Wigner transformation. Specifically, the circuit corresponds to the four-qubit operation $\exp(-i\theta/2\,\mathrm{PAR})$, with $\mathrm{PAR} = Z_1 \otimes Z_2 \otimes Z_3 \otimes Z_4$.

element inside an unstructured database. Grover's quantum search algorithm [91] can be described with the following: Given a search space of size N and no prior knowledge of the structure of the space, find the unique target element. Classically, $\mathcal{O}(N)$ queries are required to find the target element on average, while only $\mathcal{O}(\sqrt{N})$ queries are required with quantum search.

The algorithm begins with initialization of the input into a uniform superposition state:

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle \ , \tag{6.5}$$

where $|j\rangle$ are computational basis states of the N dimensional space, and j can be seen in binary form as the index of a $\log_2 N$-qubit state.

The algorithm is characterized by the *Grover iteration* where, in addition to the oracle operation corresponding to a classical query of the database, the only non-trivial operation is the so-called *Grover diffusion operator*. In practice, it is a conditional phase shift that requires the knowledge of the state of all qubits and its implementation captures most of the complexity of the algorithm apart from the oracle implementation. In abstract terms, the Grover diffusion operator applies a minus sign to a specific quantum state and takes the

form of:

$$U = \mathbf{1} - 2\,|N-1\rangle\,\langle N-1| \tag{6.6}$$

where $|N-1\rangle = |1\rangle \otimes |1\rangle \cdots \otimes |1\rangle$ in qubit language. In part of the literature, the role of the only state that acquires the minus sign is played by $|0\rangle = |0\rangle \otimes |0\rangle \cdots \otimes |0\rangle$ instead, the difference being the introduction of an extra layer of X gates and an $O(1)$ increase of circuit depth.

For practical implementations, such a multi-qubit operation needs to be decomposed in terms of gates involving at most two qubits at a time. More precisely, the key operation is the unitary version of the classical circuit that computes the logical AND of $n = \log_2 N$ bits (this is valid for the correspondence $|0\rangle$ →False and $|1\rangle$ →True). These circuits are reversibly comprised of a sequence of Toffoli gates (see next paragraph), and are characterized by gate complexity of order $\mathcal{O}(\log N)$.

To count the number of gates, we will make use of a decomposition of the Toffoli gate into 3 CNOT operations and 4 single qubit rotations, following the construction of Margolus in section VI.B of [17] and reported in Appendix 6.9.2. This construction is congruent to the canonical Toffoli gate modulo a phase shift ($|101\rangle \rightarrow -|101\rangle$). The accumulated phase is canceled before proceeding on to the next component of the algorithm. This decomposition introduces two qubit gates between the control qubits and the target qubit, which induces additional SWAP gates if the controls are not both physically adjacent to the target qubit.

As these circuits are repeatedly required for execution of any instantiation of quantum search or function inversion, they represent a wide range of quantum algorithms that show a *polynomial* speedup over the classical counterparts [31].

In our analysis we present optimized procedures making use of an additional $n-1$ ancillary qubits. These qubits enable greater algorithmic flexibility and scheduling complexity, and ultimately provide asymptotic separation in circuit depths between implementations with and without them. For reference we have included circuit depths and gate counts for a well-known [17] implementation of the same algorithm with a single ancilla, inserting SWAPs naively

when necessary.

6.9 Gate Decompositions

For normalization of analysis across the separate applications, we perform analysis with a gate set comprised of single qubit rotations, single qubit phase rotations, and the two-qubit CNOT operation. As a result, each of the gates in the considered benchmarks needs to be decomposed into this basis. Provided here is the explicit form for the *controlled phase gate*. The *Margolus Toffoli gate* can be found specifically in [17], section VI. B..

6.9.1 Controlled Phase Gate C-P_n

We wish to decompose the controlled phase operation into a sequence of single qubit gates and CNOT operations. This decomposition is provided in the bottom panel of Figure 6.13, and can be explicitly verified in matrix form as follows:

$$\begin{aligned}
\mathrm{CP}_n(q_1,q_2) &= P_{n+1}\otimes P_{n+1}\times \mathrm{CNOT}\times \mathbb{I}_2\otimes P_{n+1}^{\dagger}\times \mathrm{CNOT}\\
&= \begin{pmatrix} 1 & 0 & 0 & 0\\ 0 & e^{\frac{2\pi i}{2^{n+1}}} & 0 & 0\\ 0 & 0 & e^{\frac{2\pi i}{2^{n+1}}} & 0\\ 0 & 0 & 0 & e^{\frac{2\pi i}{2^{n}}} \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0\\ 0 & e^{\frac{2\pi i}{2^{n+1}}} & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & e^{\frac{2\pi i}{2^{n+1}}} \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0 \end{pmatrix}\\
&= \begin{pmatrix} 1 & 0 & 0 & 0\\ 0 & e^{\frac{2\pi i}{2^{n+1}}} & 0 & 0\\ 0 & 0 & e^{\frac{2\pi i}{2^{n+1}}} & 0\\ 0 & 0 & 0 & e^{\frac{2\pi i}{2^{n}}} \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0\\ 0 & e^{\frac{2\pi i}{2^{n+1}}} & 0 & 0\\ 0 & 0 & e^{\frac{2\pi i}{2^{n+1}}} & 0\\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & e^{\frac{2\pi i}{2^{n}}} \end{pmatrix},
\end{aligned}$$

where we have used $P_k = \begin{pmatrix} 1 & 0\\ 0 & e^{\frac{2\pi i}{2^k}} \end{pmatrix}$, and $\mathbb{I}_k$ as the identity matrix of dimension k. The CNOT gates act on qubits $q1$ and $q2$ with the former as the control.

6.9.2 Margolus Toffoli Gate

Following [17] we make use of the Toffoli gate decomposition as four single qubit gates and three two qubit gates, up to phase incurred on nonzero states. This phase accumulation does not have a functional effect in the Grover Diffusion circuitry we consider in this work, which enables the use of this optimized decomposition. Specifically, the Toffoli gate can be written as:

$$TOFF(q_1, q_2, q_3) = R(q_3) \times CNOT(q_2, q_3) \times R(q_3) \times CNOT(q_1, q_3) \times R^\dagger(q_3) \times CNOT(q_2, q_3) \times R^\dagger(q_3)$$

Where R is a rotation about the Y axis of $\pi/8$, and we use a convention that $TOFF(q_1, q_2, q_3)$ controls on both q_1, q_2 and targets q_3.

6.9.3 Algorithm Compositions

6.9.4 Arbitrary Two-Qubit Gate Counts

Here we show a count of the required number of two-qubit gates for each of our algorithm implementations. These figures are derived by omitting all single qubit gates and tracing over the program dependency graphs, collapsing sequential two-qubit gates acting on the same qubit operands into single gates of unspecified arbitrary definition.

As Table 6.3 shows, the QFT algorithm can absorb all of the injected SWAP operations into prior two-qubit gates, which ultimately reduces it to the original all-to-all number of two-qubit gates if they are allowed to be of arbitrary specification.

The Jordan-Wigner String results here are shown for the case of a global parity operator, which again reduces to the all-to-all case as there are no SWAP operations present. This will change when parity operators are considered that involve gaps, or involve only a small number of non-identity Pauli terms. For more connected topologies, an advantage will appear in this algorithm in these cases, as the SWAPs will not be able to be entirely absorbed by

the previous gates, which will lead to denser topologies achieving lower gate counts overall.

A unique situation occurs for the Grover Diffusion Operator. There are sets of SWAP operations that are not able to be absorbed by any prior gates within this algorithm, which results in a scaling of the number of arbitrary two qubit gates that gives an advantage to the ladder topology over that of the line. For the grid topology, the absolute number of arbitrary two-qubit gates exceeds the ladder to first order, which comes from the construction itself. The grid algorithm relies upon line subroutines that are responsible for the dominant term in the number of two-qubit gates, while a single ladder subroutine provides a scaling factor that ultimately reduces the total gate count below the line for sufficiently large n. The speedup of the algorithm is primarily due to the added parallelism, not necessarily from reducing gate count.

Arbitrary Two-Qubit Gate Counts

Architecture	QFT	Jordan-Wigner String	Grover's Diffusion Operator $n-1$ Ancilla
All-to-All	$n(n-1)/2$	$2n-3$	$6(n-1)$
Linear Nearest Neighbor	$n(n-1)/2$	$2n-3$	$10n-6\log(n)-12$
Ladder	$n(n-1)/2$	$2n-3$	$13n/2-6\log(n)-6$
2D Grid	$n(n-1)/2$	$2n-3$	$10n+\sqrt{n}-3\sqrt{n}\log n-3\log n-12$

Table 6.3: Total number of arbitrary two-qubit gates required for each algorithm implementation. Both QFT and the Jordan-Wigner String algorithms reduce to the all-to-all gate count, the former from absorbing SWAP operations into prior two-qubit operations, and the latter from the lack of SWAP operations altogether.

Algorithm 5 Linear Grover Diffusion Operation

case corresponding to $n = 2^k$
number of physical qubits is $(2n - 1)$
logical qubit x_i mapped to physical qubit q_{2i-1}

$i \leftarrow 1$
while $i \leq k$ **do**
    $m \leftarrow 2^{i-1}$
    while $m > 1$ **do**
        $j \leftarrow 0$
        while $j < 2^{k-i}$ **do Parallel**
            $s \leftarrow (1 + 2j)2^i$
            SWAP (q_{s-m}, q_{s-m+1})
            SWAP (q_{s+m}, q_{s+m-1})
            $j \leftarrow j + 1$
        end while
        $m \leftarrow m - 1$
    end while
    $j \leftarrow 0$
    while $j < 2^{k-i}$ **do Parallel**
        $s \leftarrow (1 + 2j)2^i$
        TOFF (q_{s-1}, q_{s+1}, q_s)
        $j \leftarrow j + 1$
    end while
    $i \leftarrow i + 1$
end while
Z q_n
$i \leftarrow k$
while $i \geq 1$ **do**
    $j \leftarrow 0$
    while $j < 2^{k-i}$ **do Parallel**
        $s \leftarrow (1 + 2j)2^i$
        TOFF (q_{s-1}, q_{s+1}, q_s)
        $j \leftarrow j + 1$
    end while
    $m \leftarrow 2$
    while $m \leq 2^{i-1}$ **do**
        $j \leftarrow 0$
        while $j < 2^{k-i}$ **do Parallel**
            $s \leftarrow (1 + 2j)2^i$
            SWAP (q_{s-m}, q_{s-m+1})
            SWAP (q_{s+m}, q_{s+m-1})
            $j \leftarrow j + 1$
        end while
        $m \leftarrow m + 1$
    end while
    $i \leftarrow i - 1$
end while

Algorithm 6 Ladder Quantum Fourier Transform

H (q_1)
for i in $\{1, 3, 5, \ldots n-3, n, n-2, n-4, \ldots 2\}$ **do**
    $l \leftarrow (i+1) \pmod 2$
    $j \leftarrow i$
Start Parallel
    while $j \geq 1$ **do**
        $\mathrm{CP}_{j+l-1}\ (q_j, q_{j+(-1)^l})$
        $j \leftarrow j-2$
    end while
End Parallel
    $j \leftarrow i$
Start Parallel
    while $j \geq 1$ and $j + 2(-1)^l \geq 1$ **do**
        $k \leftarrow (j+1) \pmod 2$
        $\mathrm{CP}_{j+2-2l-k}\ (q_j, q_{j+2(-1)^l})$
        $j \leftarrow j-1-2k$
    end while
    $\mathrm{H}((j+1) \pmod 2 + 1)$
End Parallel
    $j \leftarrow i$
Start Parallel
    while $j \geq 1$ and $j + 2(-1)^l \geq 1$ **do**
        $k \leftarrow (j+1) \pmod 2$
        $\mathrm{SWAP}(j, j+2(-1)^l)$
        $j \leftarrow j-1-2k$
    end while
End Parallel
end for

Algorithm 7 Ladder Grover Diffusion Operator

$i \leftarrow 1$
while $i < 2n-1$ **do Parallel**
    TOFF (q_i, q_{i+2}, q_{i+1})
    $i \leftarrow i+4$
end while
LNNGroverDiffusionOperation(second column)
$i \leftarrow 1$
while $i > 0$ **do Parallel**
    TOFF (q_i, q_{i+2}, q_{i+1})
    $i \leftarrow i+4$
end while

Algorithm 8 AND Subroutine

input: list of $2n$ qubits indexed in sequence $1, \cdots, 2n$
logical qubit x_i mapped to physical qubit q_{2i-1}

$i \leftarrow 1$
while $i \leq k$ **do**
  $m \leftarrow 2^{i-1}$
  while $m > 1$ **do**
    $j \leftarrow 0$
    while $j < 2^{k-i}$ **do Parallel**
      $s \leftarrow (1 + 2j)2^i$
      SWAP (q_{s-m}, q_{s-m+1})
      SWAP (q_{s+m}, q_{s+m-1})
      $j \leftarrow j + 1$
    end while
    $m \leftarrow m - 1$
  end while
  $j \leftarrow 0$
  if $i == k$ **then**
    SWAP (q_{n-1}, q_n)
  end if
  while $j < 2^{k-i}$ **do Parallel**
    $s \leftarrow (1 + 2j)2^i$
    TOFF (q_{s-1}, q_{s+1}, q_s)
    $j \leftarrow j + 1$
  end while
  $i \leftarrow i + 1$
end while

Algorithm 9 Ladder Jordan-Wigner String: $e^{-i\frac{\theta}{2}\sigma_z^{\otimes n}}$

$m \leftarrow \lceil n/2 \rceil$
$k \leftarrow (m + 1) \pmod 2$
Start Parallel
PARITY(Column 1,m)
PARITY(Column 2,m)
End Parallel
CNOT (q_{m-k}, q_{m+1-k})
RZ (q_{m+1-k}, θ)
CNOT (q_{m-k}, q_{m+1-k})
Start Parallel
PARITY†(Column 1, m)
PARITY†(Column 2, m)
End Parallel

Algorithm 10 Grid Jordan-Wigner String: $e^{-i\frac{\theta}{2}\sigma_z^{\otimes n}}$

m $\leftarrow \sqrt{n}\lfloor(\sqrt{n}-1)/2\rfloor$
$i \leftarrow 1$
while $i \leq \sqrt{n}$ **do Parallel**
    PARITY(Column i,q_{m+i})
end while
PARITY(Row $\lceil\sqrt{n}/2\rceil$,$q_{m+\lceil\sqrt{n}/2\rceil}$)
Rz $(q_{m+\lceil\sqrt{n}/2\rceil}, \theta)$
PARITY†(Row $\lceil\sqrt{n}/2\rceil$,$q_{m+\lceil\sqrt{n}/2\rceil}$)
$i \leftarrow 1$
while $i \leq \sqrt{n}$ **do Parallel**
    PARITY†(Column i,q_{m+i})
end while

Algorithm 11 Grid Grover Diffusion Operation

$k \leftarrow 1$
while $k \leq \sqrt{n}$ **do Parallel**
    AND(Column k)
end while
LadderGroverDiffusionOperator(center two rows)
$k \leftarrow 1$
while $k \leq \sqrt{n}$ **do Parallel**
    AND†(Column k)
end while

CHAPTER 7

EFFICIENT QUANTUM CIRCUITS FOR ACCURATE STATE PREPARATION OF SMOOTH, DIFFERENTIABLE FUNCTIONS

Effective quantum computation relies upon making good use of the exponential information capacity of a quantum machine. A large barrier to designing quantum algorithms for execution on real quantum machines is that, in general, it is intractably difficult to construct an arbitrary quantum state to high precision. Many quantum algorithms rely instead upon initializing the machine in a simple state, and evolving the state through an efficient (i.e. at most polynomial-depth) quantum algorithm. In this work, we show that there exist families of quantum states that can be prepared to high precision with circuits of linear size and depth. We focus on real-valued, smooth, differentiable functions with bounded derivatives on a domain of interest, exemplified by commonly used probability distributions. We further develop an algorithm that requires only linear classical computation time to generate accurate linear-depth circuits to prepare these states, and apply this to well-known and heavily-utilized functions including Gaussian and lognormal distributions. Our procedure rests upon the quantum state representation tool known as the *matrix product state* (MPS). By efficiently and scalably encoding an explicit amplitude function into an MPS, a high fidelity, linear-depth circuit can directly be generated. These results enable the execution of many quantum algorithms that, aside from initialization, are otherwise depth-efficient.

7.1 Introduction

Many applications depend upon initialization of a quantum register into specific states. A good example of this is the class of Monte Carlo style quantum algorithms that can compute expectation values of functions over classical probability distributions with quadratic speedup over all-classical counterparts [199, 148, 228, 151, 95]. Other examples include machine

learning classical training data sets that are used as input to a quantum machine learning algorithm [145, 184, 95, 226, 45]. Preparing the state corresponding to the data in both of these cases is in general an exponentially hard problem [177] that is often omitted from application performance analysis. However, if initialization is this difficult these applications may not be viable for quantum computing. To address this issue, we develop a technique for generating circuits that can prepare certain families of quantum states efficiently. These families of states may enable the execution of quantum algorithms that depend on classical input data on smaller, near-term machines.

We focus on smooth, differentiable, real-valued (SDR) functions with bounded derivatives, and show that they can be constructed efficiently with linear-depth quantum circuits. The primary contribution of this work is Algorithm 13, that combines a novel encoding of piecewise polynomial function approximators into matrix product state form, with existing techniques for MPS compression and low-rank MPS quantum gate extraction. The algorithm can be tuned, and different configurations have different impacts on error and scalability.

Sections 7.2 and 7.3 discuss related work and set the notation and background necessary to understand the techniques that build our algorithm. Section 7.4 shows theoretically that these types of functions display important properties, among these an exponentially decreasing von-Neumann entropy as qubits are added to the states. As a result, they have efficient MPS representations with low bond-dimension.

Section 7.5 describes our techniques, including explicit piecewise polynomial MPS function approximation, MPS arithmetic, and variational MPS approximation with low, bounded bond-dimension. Leveraging an algorithm developed in [181] that constructs a linear-depth quantum circuit from a MPS, we show that we can construct quantum circuits that prepare our desired amplitude functions with a high degree of accuracy.

Section 7.6 presents our algorithm for efficiently and scalably constructing the state preparation circuit corresponding to the functions we study. The algorithm combines techniques described in 7.5, first approximating the target function as a piecewise polynomial, encoding

each piece into an MPS, variationally compressing the MPS into one of low-rank, and extracting gates directly from this resulting state. This combination is computationally tractable, requiring $\mathcal{O}(N)$ computation, and is bottlenecked by the variational MPS compression. Stages of the algorithm can be modified as desired, for example modifying the functional form of each component of the function approximation. The result is a tunable algorithm of linear complexity in the size of the system that prepares approximated analytical quantum states with linear size and depth circuits.

Section 7.7 analyzes the performance of our algorithm targeting representative examples of SDR functions – namely Gaussian, lognormal, and Lorentzian distributions. We show numerical analysis of the accuracy of our circuits for small system sizes, and demonstrate that the techniques can prepare states with good accuracy across a range of function parameter regions.

Section 7.8 discusses how these results can be used to enable classes of quantum algorithms designed to estimate the expectation value of linear functions over probability distributions, and Section IV concludes.

7.2 Related Work

While several techniques for state preparation have been proposed, they are often expensive in either the classical compute requirements of the algorithm or the resulting quantum gate sequence. Three examples are: exact Schmidt-Decomposition [48], recursive integrable distribution encoding [90], and quantum Generative Adversarial Networks (qGAN) [233].

The most general techniques for state preparation rely primarily on iterative Schmidt-Decomposition algorithms that require exponential classical computation, scaling that is prohibitive for large states [48]. Early work from Grover [90] utilized a recursion of integral calculations that ultimately requires an exponential number of angles and classical calculations, even though the result is a potentially efficient quantum circuit [82]. Kaye and Mosca [123] developed a technique using quantum networks to construct arbitrary states,

resulting in circuits that require many-body interactions. Kitaev and Webb [129] analyzed quantum superpositions of Gaussian states. Lubasch et al. use MPS techniques to solve differential equations classically [147] and with a quantum variational technique [146]. Others have proposed qGAN learning-based approaches [233] that rely on a classically expensive combination of a variational quantum circuit and a classical neural network. These techniques construct $\mathcal{O}(poly(N))$ sized quantum circuits corresponding to the learned distributions, which have accuracy corresponding to the effectiveness of the overall learning technique.

The fundamental difference between our work and these presented methods is that our algorithm can construct accurate *linear size and depth* quantum circuits that are built from arbitrary two-qubit local gates, and only requires *linear* classical computation time to do so.

7.3 Background

This work develops a technique to construct quantum states in which each binary-indexed basis state corresponds to a coefficient that follows a specific amplitude function. We restrict our focus to smooth, differentiable, real-valued (SDR) functions that have bounded derivatives.

7.3.1 Notation

In general, for some SDR function $f(x)$ with support over some domain D we first discretize D into 2^N points for a system of N qubits, evenly placed across the domain. This uses a mapping from index k to domain values $x(k)$:

$$x(k) = a + \frac{kL}{h} \tag{7.1}$$

where $D = [a, b]$, $L = b - a$ is the width of the domain, and $h = 2^N - 1$ is the number of gridpoints. Our goal is then to construct quantum states $|\psi_{f(x)}\rangle$ defined as:

$$|\psi_{f(x)}\rangle = \bigotimes_{k=0}^{N-1} \sqrt{f(x_{S_k})}\,|S_k\rangle \tag{7.2}$$

where we will use the big-endian binary encoding of a length N binary string S written as $s_0, s_1, \cdots, s_{N-1}$ with each individual bit $s_i \in \{0, 1\}$. Big-endian here defines the mapping of binary strings to integers as:

$$k = \sum_{i=0}^{N-1} s_i \times 2^{N-1-i} \tag{7.3}$$

In this notation, we have that the function $f(x_{S_k})$ is the evaluation of the target SDR function $f(x)$ evaluated at the domain value $x(k)$ defined by the *index* k induced by the binary string S_k from equation 7.3. Written together:

$$|\psi_{f(x)}\rangle = \bigotimes_{k=0}^{N-1} \sqrt{f\left(a_0 + L\frac{\sum_{i=0}^{N-1} S_i^k \times 2^{N-1-i}}{2^N}\right)}\,|S_k\rangle \tag{7.4}$$

7.3.2 Smooth, differentiable functions

We are focusing in this work on smooth, differentiable, real-valued (SDR) functions, as these admit many properties that allow for their efficient construction inside a quantum machine. These functions have two properties that we will make use of:

- discretization accuracy increases exponentially with the number of qubits included in the system, and
- the maximum Von-Neumann entropy of the state grows much more slowly as qubits are added.

Because of the two properties derived explicitly in [82], we find a very useful scaling relationship: as the system scales up in qubit count, these functions admit efficient and accurate

representations in their low-rank approximations, while the accuracy of the encoded state continues to exponentially increase.

Examples of these types of functions include probability distributions, particularly Gaussian, lognormal, and Lorentzian distributions. We will see that the accuracy of our techniques is dependent on the smoothness of these distributions, specifically with the upper bound on the derivative of the distribution on the domain of interest: $\tilde{f}' = \max_k |f'(x_k)|$ for $k \in \{0, 2^N - 1\}$. For distributions that are relatively slowly changing, $\tilde{f}'$ is small, which leads to a more exact representation of the discretized function in a low-rank approximation.

This relationship encourages the use of efficient representations of low-rank matrix approximations, and one that is particularly suited to this task is the *matrix product state*.

7.3.3 Matrix product states

Existing literature surrounding these mathematical techniques is vast within the physics community [175, 190, 232] as well as within computational mathematics and scientific computing [161, 104, 133] where these are referred to as tensor trains. Here we describe only properties relevant to this work.

A *matrix product state* (MPS) is a collection of N tensors M_i where, in general, tensors can be any collection of values $A \in \mathbb{R}^{I_1, I_2, \cdots, I_K}$ containing K distinct indices, referred to as K-th order tensors. Scalars, vectors, and matrices are considered $0^{\text{-th}}$, 1^{st}, and 2^{nd} order tensors respectively. Within a matrix product state, we restrict ourselves to 3rd-order tensors $M_i \in \mathbb{R}^{\alpha_{i-1}, d, \alpha_i}$, where d is the number of allowed levels in our qubit model – here considered to be held at $d = 2$. Each M_i will be referred to as a *core* of the MPS. The α_j are known as *bond dimensions* [190] or *virtual dimensions* that connect the matrices together. The maximum bond dimension is denoted as $\chi = \max_i \alpha_i$, a figure can be used to upper bound the computational complexity of many routines involving the MPS. In this work, we will write each of these tensors as $M^{[i], s_i}_{\alpha_{i-1}, \alpha_i}$ to highlight the connection between the i-th tensor

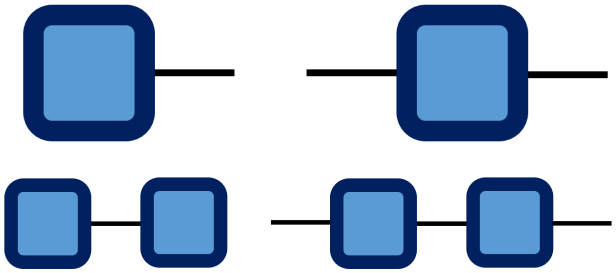


Figure 7.1: (top) Left: vector — Right: matrix. (bottom) Left: inner product — Right: matrix product

and the binary digit s_i contained in an indexing string. The MPS M_ψ can then be written as:

$$M_\psi = \sum_{\boldsymbol{\alpha},\boldsymbol{S}} M^{[1],s_1}_{\alpha_1,\alpha_2} M^{[2],s_2}_{\alpha_2,\alpha_3} ... M^{[N],s_N}_{\alpha_N,\alpha_{N+1}} \tag{7.5}$$

where the outer summations run over all bond dimensions $\boldsymbol{\alpha} = \alpha_1, \alpha_2, \cdots, \alpha_{N+1}$ and over all 2^N binary strings $\boldsymbol{S} = S_1, S_2, \cdots, S_{2^N}$. Left and right boundary bond dimensions α_1, α_{N+1} are assumed to be equal to 1.

The efficiency of this representation can be seen by looking at it as an expression of 2^N values by writing them as a matrix product. In so doing, we only need to store $\mathcal{O}(2N\chi^2)$ values, While the MPS representation is capable of representing exactly any 2^N dimensional vector by allowing χ to grow unbounded, in our study of SDR functions we will find that holding $\chi = 2$ allows for highly accurate, compact representation of SDR functions, while keeping memory cost to a modest $\mathcal{O}(N)$.

7.3.4 Tensor networks

MPS representations are just one among a family of these types of representations, where the order of all involved tensors can change. In general, these types of representations come with very useful graphical representations, and are described extensively in literature as *tensor networks* [24].

Tensors can be depicted as nodes in a graph with edges representing tensor indices. Figure

7.1 depicts single index vectors and two-index matrices in this fashion. This technique also allows us to express *tensor contractions*, which are generalizations of vector and matrix operations. Tensor contractions are operations in which one or more indices of two or more tensors are explicitly summed over, and removed from the network. As an example, the vector-vector inner product can be written as a tensor contraction of a single index, as can the matrix-matrix product.

Graphically, we describe contractions of indices by connecting the corresponding index-edge of the network together. The top of Figure 7.1 depicts the vector-vector inner product, and the bottom the matrix-matrix product. The matrix-matrix graphically can be understood by the labelling of the indices from left to right as: i, j, j, k, including two j indices for the right- and left- most indices of matrices A and B, respectively, and summing over them. Naturally these concepts generalize to any number of indices, and are used to simplify tensor networks while controlling for the complexity of the multiplications.

7.4 Theory

SDR functions have two desirable properties: as qubits are added to the system, discretization accuracy increases exponentially, while von-Neumann entropy (VNE) and therefore entanglement grows much more slowly. This can be shown by analyzing first the discretization error of a uniform gridding of the domain D on which an SDR $f(x)$ has support, followed by studying the VNE of the constructed state.

7.4.1 Discretization error

The discretization error of the encoding of an SDR function $f(x)$ into a uniform grid of 2^N points across a domain $D = [a, b] \subset \mathbb{R}$ scales as $\mathcal{O}(2^{-N})$. To see this, first let $\tilde{f}' = \max_x |f'(x)|$ be the maximum value of the derivative for $x \in D$. For a uniform gridding of the domain, we have 2^N exact values on which $f(x)$ is evaluated. Error arises when x values are sampled

that lie between any two gridpoints $(x_k, x_k + h)$ with $h = \frac{|b-a|}{2^N - 1}$, and the discretized function approximation $\hat{f}(x)$ must be interpolated. Letting the maximum evaluation error occur at $\tilde{x} = x_k + \delta$ for $\delta < 2^{-N}$, it is well known that uniform gridding and forward-difference interpolation produces first-order error linear in the step size, which in this context scales inverse-exponentially with the number of qubits in our system: $\mathcal{O}(h) = \mathcal{O}(2^{-N})$ [209].

As a result, the discretization error asymptotically halves when moving from a system of N to $N + 1$ qubits, or equivalently the function approximation accuracy doubles.

7.4.2 von-Neumann Entropy Bound

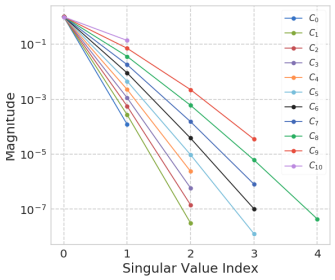
The von-Neumann Entropy (VNE) change required to add a qubit into an SDR quantum superposition decays exponentially with the system size. Defining the VNE as:

$$S(\rho) = -\mathrm{Tr}[\rho \log_2 \rho] \tag{7.6}$$

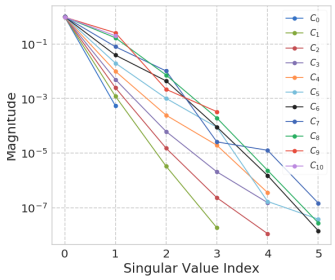
we denote the change in VNE from adding a single qubit to a quantum superposition of an SDR function $|\psi\rangle$ as $\Delta\mathrm{VNE}(|\psi\rangle)$. This was studied extensively in [82], and a bound was derived for the entropy addition as:

$$\Delta\mathrm{VNE}(|\psi\rangle) \leq \frac{L\sqrt{\tilde{f}'}}{2^{N/2-1}} = \mathcal{O}(2^{-N/2}) \tag{7.7}$$

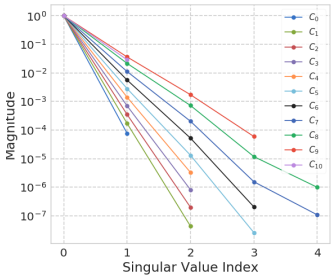
This bound on the added entropy contribution from growing a discretized SDR $f(x)$ representation, along with the reduction in the discretization error of the same order $\mathcal{O}(2^{-N})$, implies that these states scale very efficiently in their representation. One reason for this is based on the idea that VNE is a proxy for the amount of entanglement contained within a state [115, 82, 153], which would imply that growing these types of discrete function approximations requires a vanishingly small amount of entanglement.



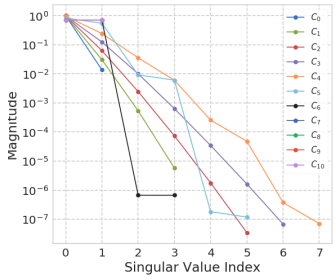
(a) Gaussian $\mu = 0$, $\sigma = 1$



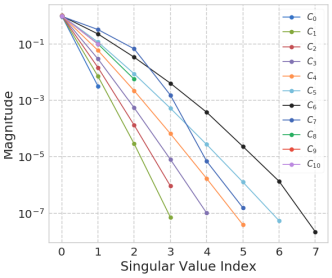
(b) Lognormal $\mu = 1$, $\sigma = 1$



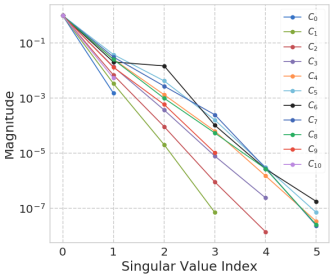
(c) Lorentzian $\mu = 0$, $\sigma = 1$



(d) Gaussian $\mu = 0$, $\sigma = 0.05$



(e) Lognormal $\mu = 1$, $\sigma = 0.05$



(f) Lorentzian $\mu = 0$, $\sigma = 0.05$

Figure 7.2: Spectral analysis of different SDR functions, discretized into a system of $N = 12$ qubits (color online)

7.4.3 Accuracy of low-χ Approximate MPS

SDR functions can be connected with MPS representations through the concept of a low-rank matrix approximation. The maximum bond dimension χ of an MPS increases with the entanglement of a system [190], and because SDR functions have a maximum VNE increase bounded inverse-exponentially by system size of the discrete approximation, these functions are accurately approximated by low-χ MPS forms.

The canonical algorithm for constructing an MPS representation of a target tensor is

Algorithm 12 MPS-SVD

Require: Target tensor A, System size N, Truncation parameter δ optional
Ensure: Approximate MPS $\overrightarrow{\mathbf{M}}$ comprised of MPS cores $M^{[1]}, \cdots, M^{[N]}$

1: $C \leftarrow A, \alpha_0 \leftarrow 1$
2: **for** $k = 1$ to $N - 1$ do **do**
3: $\quad C \leftarrow \text{reshape}(C, [\alpha_{k-1} d_k, :])$
4: $\quad U\Sigma V + E_\delta \leftarrow \delta\text{-truncated SVD}(C)$
5: $\quad M^{[k]} \leftarrow \text{reshape}(U, [\alpha_{k-1}, d_k, \alpha_k])$
6: $\quad C \leftarrow \Sigma V^T$
7: **end for**
8: $M^{[N]} \leftarrow C$
9: **return** $\overrightarrow{\mathbf{M}} = \{M^{[1]}, \cdots, M^{[N]}\}$

shown in Algorithm 12, and was designed originally in [161]. At a high level, the algorithm sweeps through the individual elements $1, 2, \cdots, N$ of the MPS, and at each site performs a singular value decomposition (SVD) of the target tensor. The resulting singular vectors are reshaped, labeled as the MPS core at this particular site.

For completeness, the *reshape* function in Algorithm 12 takes a tensor given in the first operand and resizes it into dimensions specified by the second operand. In Algorithm 12, it is used to overwrite the matrix C by a matrix with the same elements but with $\alpha_{k-1} \times d_k$ rows. The elements are read and filled in C-like order, with the last axis changing fastest.

Were we to perform a full SVD instead of δ-truncation in step 4 of Algorithm 12, the constructed MPS would be exact. If however we approximate the *unfolding matrix* C_k at step k, and leave out singular values that fall beneath some threshold δ, we incur an error $\varepsilon \leq \sqrt{\sum_{j=1}^{N} \varepsilon_j^2}$ where each ε_j is the individual truncated-SVD error given by $||E_\delta||_F$ in the algorithm, see [161] §2.2.

The optimality of the resulting MPS is given by the Eckart-Young theorem [63, 87], from which we know that the best rank k approximation to a matrix $A \in \mathbb{R}^{m \times n}$ is given by considering the first k singular values and vectors, and the error under the Frobenius norm is

equal to the total of omitted singular values:

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T \quad \rightarrow \quad ||A - A_k||_F = \sqrt{\sum_{j=k+1}^{N} \sigma_j^2}$$

This implies that the MPS constructed in δ truncated Algorithm 12 is an optimal MPS for the specified δ; the MPS core formed by approximating unfolding matrix C_k in step 4 is optimal.

With these conditions, we can estimate the accuracy of bounded-χ approximate MPS representations of SDR functions. Connecting the δ-truncated SVD error with the error from Algorithm 12, we see that the approximate MPS $\overrightarrow{\mathbf{M}}$ with maximum bond dimension bounded at χ has Frobenius error upper bounded by the sum of all squared omitted singular values of the unfolding matrices:

$$||A - \overrightarrow{\mathbf{M}}||_F^2 \leq \sum_{j=1}^{N} \varepsilon_j^2 = \sum_{j=1}^{N} \left(\sum_{k=\chi+1}^{\dim(C_j)} \sigma_k^2(C_j) \right) \tag{7.8}$$

Based on this, we conjecture and show empirically that the spectra of unfolding matrices decays exponentially for discretized SDR functions, potentially because the entropy grows inverse-exponentially as qubits are added. This implies the existence of accurate low χ MPS approximations, and we show high-accuracy approximations even for $\chi = 2$. Unfolding matrix spectra are shown in Figure 7.2, while numerical evidence supporting the exponential decay of equation (7.9) is shown in Figure 7.3.

We can estimate the accuracy of $\chi = 2$ MPS approximations to SDR functions by modeling the spectra of unfolding matrices with exponential decay. Allowing for each C_j unfolding matrix to follow a distinct exponential decay, we can formulate an exponential univariate multiple regression with the model shown in equation (7.9).

$$\sigma_k^j = \alpha_j e^{-\beta_j k} \tag{7.9}$$

We have a two-parameter univariate exponential decay model for the spectrum of each C_j, where the j-th unfolding matrix spectrum is characterized by empirically fit parameters α_j, β_j. Under this model, we can calculate the normalized upper bound of the error of an MPS approximation with bounded bond-dimension χ, shown in equation (7.10), where we have assumed for simplicity of analytics that all α, β terms are approximately equal. This allows us to estimate that for $\chi = 2$ and all $\beta \geq 1.152$, there will exist an MPS representation with greater than 99% normalized accuracy.

$$\begin{aligned}\frac{||A - \overrightarrow{\mathbf{M}}||_F^2}{||A||_F^2} &\leq \frac{\sum_{j=1}^{N}\left(\sum_{k=\chi+1}^{\dim(C_j)}(\alpha_j e^{-\beta_j k})^2\right)}{\sum_{j'=1}^{N}\left(\sum_{k'=1}^{\dim(C_{j'})}(\alpha_{j'} e^{-\beta_{j'} k})^2\right)} \\ &= \frac{e^{\beta(N-2)}(e^{2\beta} - e^{-2\beta})}{e^{\beta N} - e^{-\beta N}} \\ &= e^{\beta(N-\chi)}\text{csch}(\beta N)\sinh(\chi\beta) \end{aligned} \tag{7.10}$$

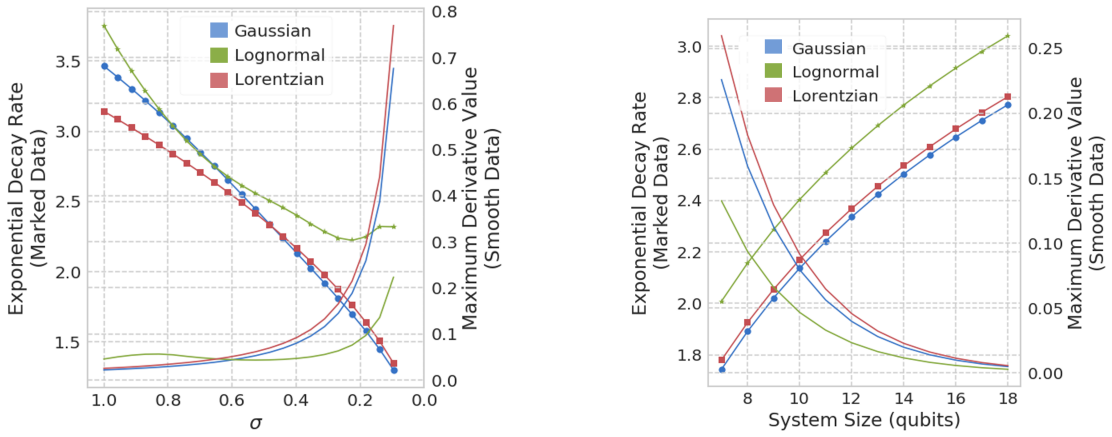
7.4.4 Numerical SDR Spectral Analysis

The validity of a singular value decay rate following the model of equation (7.9) can be numerically estimated. Empirically, we find that the spectra are well modeled by this form, and estimated β values are often larger than this threshold. We highlight univariate Gaussian, lognormal, and Lorentzian distributions, as they are representative distributions commonly used in applications, and each is discretized across a bounded support interval. The probability density functions of these distributions are:

$$p_G(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{(-(x-\mu)^2)/2\sigma^2} \tag{7.11}$$

$$p_{Ln}(x;\mu,\sigma) = \frac{1}{x} p_G(\log(x)) \tag{7.12}$$

$$p_L(x;\mu,\sigma) = \frac{\sigma}{2\pi}\frac{1}{(x-\mu)^2 + \sigma^2} \tag{7.13}$$



(a) (b)

Figure 7.3: Marked data indicates empirically fit exponential decay rates across all unfolding matrix spectra for each SDR distribution, plotted against (a) decreasing standard deviation and (b) increasing system size for fixed $\sigma = 0.4$. Smooth lined data indicates the maximum derivative achieved for each of the distributions. For all distributions, as the standard deviation decreases, the maximum derivative increases while the exponential decay rate of the singular values decreases. The lognormal distribution qualitatively changes shape around $\sigma \leq 0.2$, leading to a change in the maximum derivative resulting in a phase change of the decay rate of the singular values. We see in (b) an increase in the rate of exponential decay as systems increase in size, indicating that $\chi = 2$ MPS approximators remain effective as systems are increased in size.

Figure 7.2 shows the unfolding matrix spectra for Gaussian, lognormal, and Lorentzian distributions with differing standard deviations. Loosely, as the standard deviations decrease for these functions the maximum derivative of the functions obtained on the supported domains increases: states with low σ have higher maximum derivatives than their high-σ counterparts. This likely contributes to the fact that low-rank MPS approximations have difficulty capturing these highly localized features.

The joint exponential decay rate is found by fitting the model of equation (7.9) to all of the composite spectra. We find that for these distributions, the decay rates are above 1.152 for all but the lowest standard deviation Gaussian distributions. Results are shown in Figure 7.3. As these distributions become tighter with smaller standard deviation, these functions gain larger and larger derivatives through the supported domain, which likely prevents low-rank approximations from capturing these highly local features completely. Their unfolding matrix

spectra decay slower and slower as well. Empirically, good $\chi = 2$ MPS representations of these distributions can be formed with greater than 99% accuracy as is predicted by our analytical model, so long as the standard deviations are moderately valued, holding the discretization domain constant. We also see that as systems increase in size, the value of the maximum derivative decreases, and the exponential decay rates actually increase. This indicates that $\chi = 2$ MPS likely remain good if not better approximations for states discretized over large systems.

7.5 Techniques

The core of our algorithm rests on four main techniques: piecewise polynomial function approximation, MPS arithmetic, iterative MPS compression, and quantum gate extraction from MPS representations.

7.5.1 Piecewise polynomial function approximation MPS

In many cases, matrix product states are used to encode low-rank approximations to data which do not have a known analytical form. In these cases MPS forms can be constructed using exact construction as in Algorithm 12. They can also be approximately constructed using algorithms that subsample a portion of the domain and interpolate [160], extract dominant singular values exactly [132, 56], or estimate the dominant singular values potentially with randomized algorithms [109, 23, 43]. Recent work [55] applies these techniques to develop a method in this fashion for sampling potentially exotic multivariate probability distributions.

In our case, we are presented with an analytical form of the state we are constructing. Many functions with analytic forms can be exactly written down in a matrix product state, as shown in [159]. However, a technique to do so requires that these functions are *rank-r*

separable. This means that these functions can be written as:

$$f(x+y) = \sum_{\alpha=1}^{r} u_\alpha(x) v_\alpha(y) \tag{7.14}$$

for some fixed r value. Unfortunately, this property does not hold for many probability density functions. It does hold however for degree-p polynomials:

$$f(x) = \sum_{k}^{p} a_k x^p \tag{7.15}$$

An explicit form of discretized functions of the form (7.15) can be written as:

$$f\left(\sum_k t_k\right) = M^{[1],s_0}_{\alpha_1,\alpha_2} M^{[2],s_1}_{\alpha_2,\alpha_3} \ldots M^{[N],s_N}_{\alpha_N,\alpha_{N+1}} \tag{7.16}$$

$$\begin{aligned} \phi_s(x) &= \sum_{k=s}^{p} a_k \binom{k}{s} x^{k-s} \\ M^{[k],t_k}_{i,j} &= \begin{cases} \binom{i}{i-j} x^{i-j} & i \geq j \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{7.17}$$

where we have for the first and last tensors:

$$\begin{aligned} M^{[1],t_1} &= \left(\phi_0(t_1), \phi_1(t_1), \cdots, \phi_p(t_1)\right) \\ M^{[N],t_N} &= \left(1, t_N^1, t_N^2, \cdots, t_N^p\right) \end{aligned}$$

These MPS forms have bounded $\chi \leq p+1$, see [159], §6.

Motivated by this, we derive novel MPS forms of *piecewise* polynomial (PP) functions with bounded support on a subregion of the gridded domain. Specifically, for a domain $D = [a, b]$ and subdomain a', b' such that $a < a' < b' < b$, a polynomial function with support

on domain $D' = [a', b']$ can be written as:

$$f(x) = \begin{cases} \sum_k^p a_k x^k & a' \leq x \leq b' \\ 0 & \text{otherwise} \end{cases} \tag{7.18}$$

Based on a binary encoding of the original domain, subdivide the domain into a set defined by *support-bit* k ordered from the left. This creates 2^k different regions, each defined as:

$$D_j = \left[a + j\frac{2^k L}{h}, a + j\frac{2^k L}{h} + \frac{2^{N-j} L}{h} \right) \tag{7.19}$$

Here we use the encoding provided in equation (7.1), where the j^{th} polynomial is supported on the region indexed by j, and assert that the last region is inclusively bounded. This creates a uniform gridding of the domain into 2^k evenly spaced partitions, each of which then supports a single function approximating polynomial.

The j^{th} polynomial in a piecewise approximation over a support-bit k divided domain can be referred to as $g_j(x)$, and can be written into an MPS as in form (7.16), with explicit zeroing out of the tensors that correspond to domain values outside support. To do this, we write out $\boldsymbol{b_j} = \{b_1, b_2, \cdots, b_k\}$ a binary representation of the index j using exactly k bits. Then, for each tensor i from $1 \cdots N$, we zero out the component of the tensor that is unsupported in the binary encoding of the domain. Explicitly, for all $1 \leq i \leq k$:

$$M^{[i],t_i} : \begin{cases} M^{[i],t_i=1} = \mathbf{0}^{\boldsymbol{p+1} \times \boldsymbol{p+1}} & \text{if } b_i = 0 \\ M^{[i],t_i=0} = \mathbf{0}^{\boldsymbol{p+1} \times \boldsymbol{p+1}} & \text{if } b_i = 1 \end{cases} \tag{7.20}$$

where the remaining $M^{[i],t_i}$ for $i > k$ are all unchanged. Equation (7.20) enforces that the polynomial $f_j(x)$ is zeroed out for any domain value that lies outside of the range D_j.

With this explicit form of a bounded-support polynomial, we can write the total MPS of

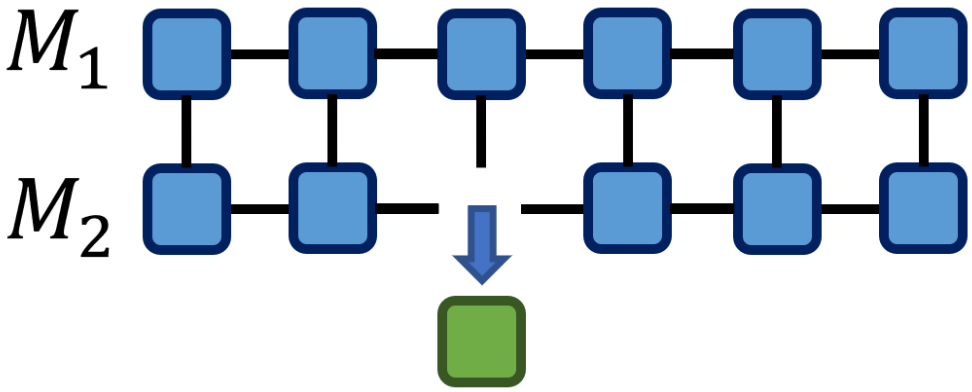


Figure 7.4: Evaluating the partial derivative of the overlap between two matrix product states, at site 2, as zero-indexed from the left. Full contraction of $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$ is completed, omitting site 2. The resulting tensor system is solved for the optimal site-2 tensor that maximizes value of the overlap, with normalization constraints. This new tensor replaces the original site-2 tensor.

a piecewise polynomial function used as a function approximator. A piecewise polynomial approximation function $g(x)$ with support on domain D is constructed by subdividing $D = \{D_1, D_2, \cdots, D_{2^k}\}$ into 2^k subregions, and fitting 2^k possibly-distinct piecewise polynomials to the function $g(x)$, with each polynomial supported on a single subregion. Together, this forms a piecewise polynomial approximation $\tilde{g}(x)$ to $g(x)$:

$$\tilde{g}(x) = \begin{cases} g_1(x) & x \in D_1 \\ \dots & \\ g_k(x) & x \in D_k \end{cases} \tag{7.21}$$

Here we do not require the functions be continuous at the endpoints of subregions.

With a function approximation written down this way, we can iteratively construct a series of 2^k MPS forms corresponding to each of the approximating polynomials.

7.5.2 Matrix product state arithmetic

Once we form 2^k MPS forms, we can combine them using the arithmetic properties of matrix product states. Specifically, two matrix product states can be added together as:

$$\boldsymbol{M_1} + \boldsymbol{M_2} = \boldsymbol{M_3} = M_3^{[1],s_1} M_3^{[2],s_2} ... M_3^{[N],s_N}$$

where each $M_3^{[i]}$ term is a block diagonalization of the corresponding terms in each summand:

$$M_3^{[i],s_i} = \text{diag}\left(M_1^{[i],s_i}, M_2^{[i],s_i} \right)$$

appropriately adjusting for single-dimensional row and column vector endpoint cores. Upon contraction, the result is the addition of the two encoded functions:

$$\begin{aligned} \boldsymbol{M_3}(x) &= \sum_{\boldsymbol{\alpha_1}} \prod_{i=1}^{N} M_{1,\alpha_i,\alpha_{i+1}}^{[i],s_i} + \sum_{\boldsymbol{\alpha_1}} \prod_{i=1}^{N} M_{2,\alpha_i,\alpha_{i+1}}^{[i],s_i} \\ &= \boldsymbol{M_1}(x) + \boldsymbol{M_2}(x) \end{aligned}$$

Using this property, we can combine the 2^k MPS forms defined in a piecewise polynomial function approximation. Each of the constituent MPS forms have a maximum bond-dimension defined by the degree of the encoded polynomial: $\chi_j \leq p_j + 1$. MPS addition in this way grows the rank of the resulting MPS by exactly the ranks of the constituent MPS. Because of this, the MPS formed by the addition of 2^k degree-p piecewise polynomial MPS forms has rank $\chi_{\text{total}} = 2^k(p+1)$.

7.5.3 Iterative MPS Compression

Large-χ matrix product states can be compressed into a lower-χ MPS by an iterative method, focusing on a single core at a time. Following [190], the optimal approach to compress an MPS $\boldsymbol{M_1}$ of rank χ_1 into $\boldsymbol{M_2}$ of rank $\chi_2 < \chi_1$ is to begin with an ansatz for $\boldsymbol{M_2}$ of rank

Algorithm 13 SDR Function Encoding

Require: Target SDR function $f(x)$, System size N, Support-bit k, Domain $D = [a, b]$, Accuracy parameter ε optional
Ensure: Quantum circuit $U = g_1 g_2 \cdots g_k$ such that
$U|0\rangle = |\psi_{f(x)}\rangle$

$R \leftarrow \{D_1, D_2, \cdots, D_{2^k}\}$ ▷ binary subdivided domain
$\tilde{g}(x) \leftarrow \{g_j(x) \mid \forall j \in [1, 2^k]\}$ ▷ PP approx of $f(x)$
for $j = 1$ to 2^k do **do**
    $M_j \leftarrow \text{MPS}(g_j(x))$ ▷ MPS encoding of g_j §7.5.1
end for
$M_T \leftarrow \sum_k M_k$ ▷ MPS summation §7.5.2
$\tilde{M}_T \leftarrow \text{IterCompress}(M_t)$ ▷ MPS compression §7.5.3
$G \leftarrow \text{GateExtraction}(\tilde{M}_T)$ ▷ MPS to q-gates §7.5.4
return $G = g_1, g_2, \cdots, g_k$

χ_2, and change each core $M_2^{[k],s_k}$ iteratively. For core k, the update rule follows from the gradient of the overlap between both states, calculated with respect to core k. This gradient is of the form:

$$\frac{\partial \langle \boldsymbol{M_1} | \boldsymbol{M_2} \rangle}{\partial M_2^{[k]}} = \sum_{\boldsymbol{\alpha}, \boldsymbol{S}} \left[\left(\prod_{i \in \mathcal{I}} M_1^{[i],s_i} \right)^\dagger \prod_{j \in \mathcal{I}/\{k\}} M_2^{[j],s_j} \right] \tag{7.22}$$

which corresponds to a full pairwise contraction of the conjugate of each core in $\boldsymbol{M_1}$ with the corresponding core of $\boldsymbol{M_2}$, *omitting* the k-th core in $\boldsymbol{M_2}$. As such, above the index sets $\mathcal{I}$ are defined as the ordered set $\{N, N-1, \cdots, 1\}$. In graphical notation this is simplified as shown in Figure 7.4.

The iterative compression algorithm evaluates equation (7.22) at each site k, and calculates the optimal core k to replace the existing k-th core. This calculation corresponds to solving a $\left(\chi_2^2 \times \chi_2^2\right)$ dimensional linear system, and using factorizations presented in [190] can be performed in $\mathcal{O}(\chi_2^3)$ time. In practice then, this algorithm can be computationally bounded at $\mathcal{O}(N\chi_2^3)$ time, where a fixed number of sweeps and solutions are performed over all N cores.

7.5.4 *Accurate linear-depth circuits*

Once a suitable matrix product state has been constructed, a technique developed recently in [181] can be used to directly convert the MPS into a set of one and two-qubit gates. This is performed by calculating the "matrix product disentangler" $\hat{U}$ with the property that it acts on the state $|\psi\rangle$ encoded by the MPS and creates the vacuum state.

The procedure for construction of this unitary operator acts on each MPS core $1 < k < N$ and forms two-qubit gate $G^{[k]}$:

$$G^{[k]}_{0 j s_k l} = M^{[k],s_k}_{j,l}$$

This forms half of the required elements for the two qubit operator $G^{[k]}$, and the other half are chosen by selecting $(d^2 - d)$ orthonormal vectors in the kernel of $M^{[k]}$. This fills in the two qubit operator, and results in a unitary gate. Sites 1 and N are filled in similarly, adjusting for specific dimensional constraints. The result is a set of $N + 1$ unitary quantum operations, N of which are two qubit gates, that form a serialized circuit of depth linear in system size: $N + 1$. Details of the procedure are shown clearly in [181]. These circuits can be decomposed into canonical gates using $7N + 1$ single qubit gates and $3N$ two qubit gates, at a depth of $\sim 6N$, utilizing standard decompositions [48]. Recent work [146] has generalized these types of techniques to higher χ MPS, which directly enable further generalization of our Algorithm 13.

7.6 Algorithm

All of the techniques from Section 7.5 are combined into Algorithm 13, which has a time complexity of $\mathcal{O}(N\chi^3) = \mathcal{O}(8N)$ for $\chi = 2$ MPS approximations. Proving this requires analyzing each component of the algorithm.

Procedures in lines 1, 4, and 6 are constant time components. Each of the MPS encodings of line 4 can be performed in parallel, as they are each independent and are a constant time

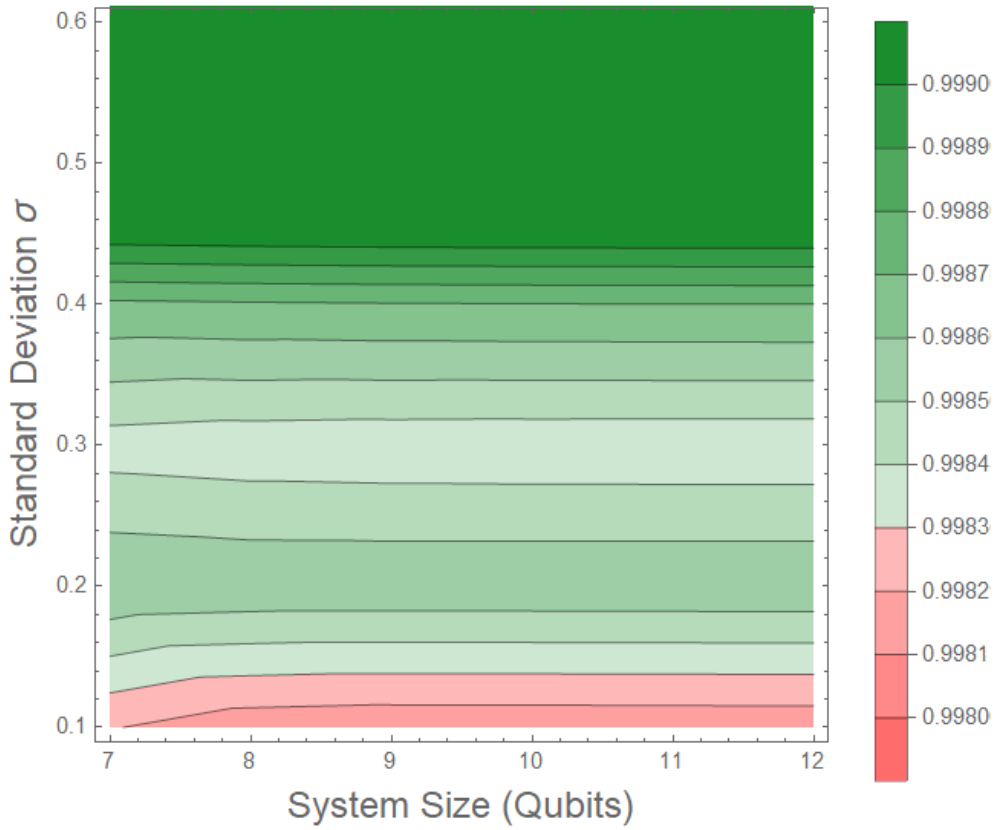


Figure 7.5: Scaling of circuit fidelities with the standard deviation of the target probability distribution, across low and intermediate σ. All circuits construct states with greater than 99% fidelity through $\sigma \geq 0.1$, and exceed 99.9% for all system sizes and all distributions above $\sigma = 0.44$. Below, circuit fidelities decay with the ability to approximate the distributions with low rank $\chi = 2$ MPS forms, in agreement with predictions from equation (7.10). In each distribution, we set $\mu = 1$ and bound support on domain $D = [0, 2]$, with $D = [\varepsilon, 5]$ for the lognormal distribution to capture relevant features. The $\sigma = 0.3$ region defines qualitative changes in the Gaussian and lognormal distributions that make approximation by $\chi = 2$ MPS more difficult.

operations, following the explicit analytical form prescribed in equation (7.16) and truncating with equation (7.20). This is an important component of the algorithm, and the number of regions 2^k is a constant that often provides sufficient accuracy when chosen to be small (e.g. 8). MPS summation in line 6 is also constant-time as it is reorganizes the tensors into block diagonalizations of the constituent piecewise-supported MPS.

Constructing the piecewise approximation of the function in line 2 has complexity that reflects the method used to do the approximating. Each subregion $D_j \in R$ is independent,

so all 2^k approximations can be performed in parallel. A single approximation over region j by a bounded degree-p polynomial can be performed with complexity that scales with the number of distinctly sampled points on each subregion. For a gridding of domain D_j into L points, the least squares polynomial regression can be performed in $\mathcal{O}(p^2 L)$, where both p and L are constants chosen and customizable to particular target functions.

The iterative MPS compression of line 7 is the dominant contributing factor to the complexity of the algorithm, requiring $\mathcal{O}(N\chi^3)$ where we are targeting states with $\chi = 2$, reducing this to $\mathcal{O}(8N)$. This complexity bound arises as the compression can be simplified into computing the optimal single-tensor that solves N distinct overlaps between a $\chi_1 = p+1$ state and the optimized $\chi_2 = 2$ state, each of which amounts to solving a $(\chi_2^3 \times \chi_2^3)$ linear system, after accounting for useful properties afforded by the MPS representation [190].

Gate extraction [181] is also a linear time operation, requiring the inversion of N $\chi_2 \times \chi_2$ matrices to complete each quantum gate. This can be done naively in $\mathcal{O}(\chi_2^3)$ time, and at best $\mathcal{O}(\chi_2^{2.373})$ [40]. Once again, all N matrices can be inverted in parallel, reducing this complexity to $\mathcal{O}(N\chi^{2.373})$.

Altogether, iterative MPS compression is the dominant computational cost of the algorithm when simple function approximation and optimized matrix inversion techniques are used, resulting in a time complexity for $\chi = 2$ targets of $\mathcal{O}(8N)$.

7.7 Analysis and Results

7.7.1 Performance analysis and Numerical Results

To evaluate the accuracy of our technique, for small systems we explicitly compare the constructed state with the target. This is done with the *fidelity* measure $\mathcal{F}$ [153], defined as:

$$\mathcal{F}(|\phi\rangle, |\psi\rangle) = \mathrm{Tr}\sqrt{\rho^{1/2}\sigma\rho^{1/2}} = |\langle\phi|\psi\rangle|^2 \tag{7.23}$$

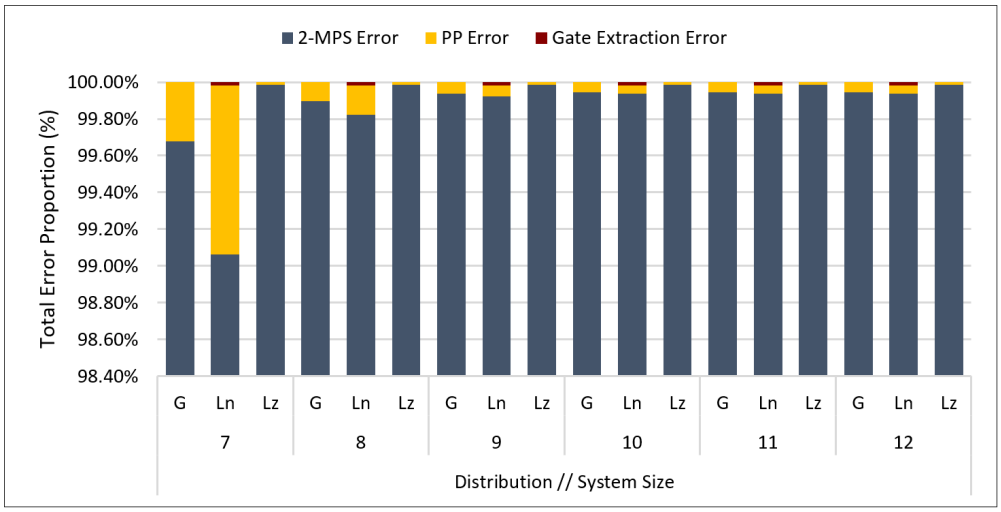


Figure 7.6: Decomposing the total error of each state construction by source. G, Ln, and Lz correspond to Gaussian, lognormal, and Lorentzian distributions, respectively. Configurations correspond exactly to those constructed in Figure 7.5, for fixed $\sigma = 0.1$.

where $\rho = |\phi\rangle \langle\phi|$ and $\sigma = |\psi\rangle \langle\psi|$, and right equality holding for ϕ, ψ pure states. Using this measure, we can estimate how exactly the constructed states match the targets.

Figure 7.5 depicts the fidelities of circuits generated by Algorithm 13 plotted against the standard deviation of the targeted SDR distribution, in different regimes. We expect to see that as σ approaches 0 fidelity should decrease as the MPS is no longer able to accurately capture the large maximum upper bound on the derivative of the distribution. In Figure 7.5 this occurs for $\sigma \sim 0.12$, closely matching predictions by the spectral analytical modeling in equation (7.10). This can be attributed to equation (7.7), where the added entropy of the state grows with the maximum derivative of the SDR function $\tilde{f}'$. For very low standard deviation states, this large effective constant dominates equation (7.7), and we find low-χ MPS states unable to accurately represent the function. Additionally, we find a region $\sigma \approx 0.3$ in which accuracy slightly decays as well. This reflects a fluctuation of the rank of the distributions in this region, beneath the upper bounds set by the derivatives.

7.7.2 Error Analysis

Error arises in three places within Algorithm 13: the piecewise polynomial approximation $\tilde{g}(x)$, the compression of the total MPS, and gate extraction. The dominant error comes from **MPS compression**, followed by the approximation error of the piecewise polynomial function (PP error). Gate extraction error is negligable. Figure 7.6 displays the decomposition of the total error in each constructed distribution and attributes each to its source. Among all constructed cases, MPS error contributes to 99.9% on average, while PP and gate extraction error account for 0.11% and 0.01% on average, respectively. For the lognormal distribution, PP error contributes more significantly than in any other distribution, accounting for 0.92% for $N = 7$ qubit constructions.

The polynomial function approximation error can be decreased by increasing the degree of the fit polynomial. Doing so comes at the cost of increasing constants in the computational complexity of the entire procedure, and for large values relative to the system size the practical implementation complexity may begin to be affected. Figure 7.7 displays sensitivity of Algorithm 13 to the order of polynomial approximator used, studied for a single instantiation of each SDR function: $\sigma = 0.1$. Accuracy increases with approximation degree, with diminishing returns beginning at the second order. Even for this difficult set of functions, cubic polynomials are able to construct the states with over 99% accuracy, and increasing to 5^{th} order polynomials increases the fidelity up to $99.8\%, 99.91\%$, and 99.95% for each of the Gaussian, lognormal, and Lorentzian distributions, respectively.

7.7.3 Optimality Ratios

The scalability of the technique rests on the conjecture that $\chi = 2$ MPS representations remain good approximators for the target functions as the system size scales up. Empirically this can be estimated by tracing the fidelity of the SVD $\chi = 2$ MPS representations to the SDR function and scaling up system size. The SVD $\chi = 2$ MPS can be determined with Algorithm 12, which does not include function approximation. Figure 7.8 shows these curves

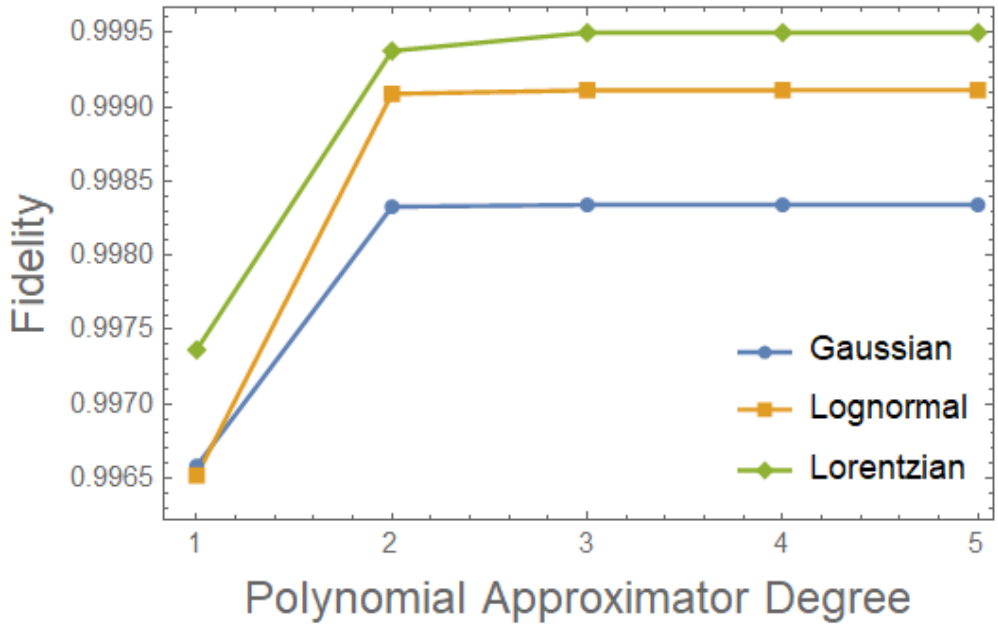
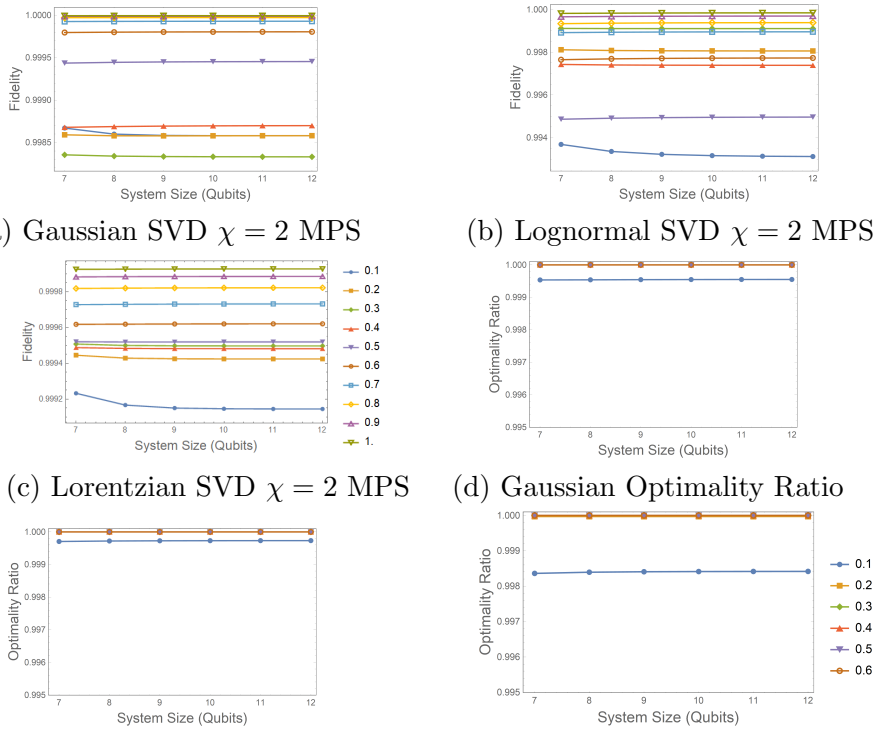


Figure 7.7: Accuracy of Algorithm 13 sweeping through degrees of polynomial approximators, for a fixed sample size within subregions and for distributions with fixed $\sigma = 0.3$. Diminishing returns are seen for polynomials higher than order 2, which is a property of these specific functions.

for a selection of standard deviations, and presents numerical support for the claim that these $\chi = 2$ approximations remain consistently accurate for larger and larger systems. As a result, the gate extraction component of Algorithm 13 remains unaffected by an increase in the size of the system overall.

The main factor in the accuracy of Algorithm 13 is then the construction of the approximate $\chi = 2$ MPS. In order to estimate how accurately the constructed state matches the optimal, we can use an *optimality ratio* metric, defined by the ratio of the fidelity of our generated circuits to the fidelity of approximation free MPS generated circuits. These are shown in Figure 7.8 (d-f).

For lower standard deviation states the absolute fidelity of the state constructed by Algorithm 13 remains high and the optimality gap remains constant or slightly improves. This is strong evidence that accuracy will scale well with larger system discretizations.



(a) Gaussian SVD $\chi = 2$ MPS (b) Lognormal SVD $\chi = 2$ MPS

(c) Lorentzian SVD $\chi = 2$ MPS (d) Gaussian Optimality Ratio

(e) Lognormal Optimality Ratio (f) Lorentzian Optimality Ratio

Figure 7.8: (a-c) Scaling of SVD $\chi = 2$ MPS fidelities, as generated by the exact Algorithm 12 for (a) Gaussian, (b) lognormal, and (c) Lorentzian distributions. (d-f) Scaling of the optimality ratio of the approximation-free low-rank MPS as generated by the exact Algorithm 12 against the MPS generated by Algorithm 13 for (d) Gaussian, (e) lognormal, and (f) Lorentzian distributions.

7.8 Applications

A primary application for this procedure is any Monte Carlo style quantum algorithm that estimates the value of an observable evaluated on classical probability distributions. Monte Carlo methods have been shown to have quadratic speedup in these domains [151], and to demonstrate this we discuss the Amplitude Estimation based financial derivatives pricing algorithm.

7.8.1 Application to Monte Carlo Methods

Many algorithms have been proposed in the risk analysis and financial derivatives pricing context [199, 228, 183, 182, 158, 180]. Many of these are based around the same principle:

after solving a variation of the Black-Scholes equation that dictates the movement of an asset, encode this solution as a probability distribution into a quantum state. Once this is complete, then a linear function is computed on this state, which represents a superposition over all values of this linear function, weighted by probability. This computes an expected value of this operator, which in many cases is formulated specifically to encode the price of a financial derivative of the asset. The expected value is evaluated using Amplitude Estimation [32], with error and convergence that scales as:

$$T = \mathcal{O}(T_{\text{prep}}/\varepsilon_{\text{sampling}})$$

Classical Monte Carlo techniques for the same function scale as $\mathcal{O}(1/\varepsilon^2)$, giving the quantum algorithm a quadratic speedup. Clearly though, there is dependence on T_{prep}, or the time required to encode this distribution. With Algorithm 13, this can be done in linear time with tuneable accuracy.

7.8.2 Extensions and Future Work

One benefit of this work is that it extends to any real-valued functions that are well-approximated by a set of piecewise low-order polynomials. This work thus extends to cover classical input data sets, so long as the data can be well approximated with an analytical generating function. One example of this is image data, which often can be approximated or interpolated into an approximate analytical form [126, 134, 107, 168]. This in theory would allow for quantum states corresponding to image data to be efficiently constructed, given a method for multivariate MPS encoding.

7.9 Conclusion

In this work we develop an algorithm to prepare quantum states corresponding to smooth, differentiable, real-valued functions. The algorithm constructs a linear-depth circuit of arbitrary

two-qubit quantum gates, and does so only requiring linear computation time. Evaluating the accuracy of this technique empirically on commonly used probability distributions shows that high degrees of accuracy are able to be obtained even for the lowest standard deviation target functions. These techniques require computation that scales linearly with system size, and there is no evidence that accuracy decays as systems increase in size, showing promise for the scaling of these techniques to much larger systems.

CHAPTER 8

PROVING THE EFFICIENCIES OF SDR FUNCTION ENCODING

This section provides rigorous analytical bounds on the efficiencies of an MPS based encoding method for the construction of state preparation circuits of SDR functions. The work classifies the ability for low-rank MPS to generate high-fidelity circuits for SDR functions, based on properties of the SDR function shape and rates of change. The analytics complement the existing numerical work by providing bounds on the accuracy of the techniques as systems are scaled up in sizes prohibitive of direct fidelity calculation.

8.1 Introduction

The promise of quantum computation is the ability to solve problems exponentially faster than we can today with classical computers. However, the performance of many quantum algorithms depends on the ability to load classical data efficiently and accurately into quantum states. For example, this capability is necessary for quantum computers to be viable for performing machine learning with large classical training data sets [95, 45, 145, 184, 226] and Monte Carlo calculations that compute expectation values of functions over classical probability distributions [95, 228, 199, 151]. In both of these cases, preparing the state corresponding to the data is vital to preserving the quantum speedup present in the remainder of the algorithm. While in general, state preparation is an exponentially hard problem [177], it has previously been shown empirically that certain families of quantum states can be prepared efficiently and with high precision using linear-depth circuits, and algorithms have been developed that require only linear compute time to generate these linear-depth circuits for real-valued smooth, differentiable (SDR) functions [102]. The practical utility of this approach depends upon whether it is scalable to large qubit systems.

In this paper, we prove analytical upper bounds on the entanglement requirements

for quantum state superpositions of discretized SDR functions, and thereby assess and demonstrate scalability of state preparation encoding methods using matrix product state (MPS) data structures. These have been shown to be powerful for simulating correlated one-dimensional systems [219, 220], and bounds have been established showing domains over which MPS encodings remain efficient data structures [191]. In the present work, we prove that superpositions corresponding to discretized SDR functions require entanglement that grows logarithmically in system size, thereby also showing that they can be accurately simulated with matrix product states as is shown in [191].

8.2 Results

Theorem Lemma Corollary

The proof structure is quite simple. We construct a polynomial approximation to the target quantum stateand bound the entropy of an encoding of this polynomial into a corresponding MPS. The entropy of this encoding is upper bounded by the bond-dimension of the MPS, which is used to complete the proof. The main result is stated and proved in Theorem 8.2.5, and relies on several lemmas which are stated first.

The first lemma bounds the pointwise approximation accuracy of polynomial function approximations in terms of the domain of the function D, the derivative of the function γ, and the polynomial approximation degree k.

Lemma 8.2.1. *Let $D \subseteq \mathbb{R}$ be a closed bounded interval and assume that f is a function that satisfies, for some $C_f, \gamma_f \geq 0$*

$$\|f^{(n)}\|_\infty \leq C_f \gamma_f^n n! \quad \forall n \in \mathbb{N}_0 \tag{8.1}$$

then a polynomial g with support over the same domain D requires a degree p to achieve

pointwise accuracy $\|f - g\|_\infty = \varepsilon$ *as:*

$$p = \log_\alpha \left(1/\varepsilon\right) + C \tag{8.2}$$

for constant C *and* $\alpha = 1 + \frac{2}{\gamma_f |D|}$.

Proof. Lemma 3.13 of [29] shows that for the appropriate domains D and functions f, we have that, for all $k \in \mathbb{N}_0$

$$\min_{g \in \mathcal{P}_k} \|f - g\|_{\infty, D} \leq \frac{C_f 4e(1 + \gamma_f |D|)(k+1)}{\left(1 + \frac{2}{\gamma_f |D|}\right)^{(k+1)}} \tag{8.3}$$

where the family $\mathcal{P}_k$ is the family of polynomials of degree k. This is used in [89] to prove the statement above. □

The next lemma is a simple upper bound on the ℓ_1 norm of quantum states, or normalized vectors $\boldsymbol{v} \in \mathbb{C}$.

Lemma 8.2.2. *Let* f *be a discretized SDR function over a closed bounded interval* $D \subseteq \mathbb{R}$ *normalized such that* $\|f\|_2 = 1$. *Then we have:*

$$\max_f \|f\|_1 \leq 2^{N/2} \tag{8.4}$$

This bound is tight, saturated by the uniform distribution as: $f_i = 2^{-N}$ *for all entries.*

Proof. Apply the Cauchy-Schwarz inequality and use the all-ones vector $\mathbf{1} = \{1\}_{i=1}^{2^N}$:

$$\|f\|_1 = \|f \cdot \mathbf{1}\|_1 \leq \|f\|_2 \|\mathbf{1}\|_2 \tag{8.5}$$

$$= 1 \left(\sum_i^{2^N} 1 \right)^{1/2} = 2^{N/2} \tag{8.6}$$

We saturate this with the uniform distribution u with $u_i = 2^{-N/2}$ for all i as $\|u\|_1 = 2^N (2^{-N/2}) = 2^{N/2}$. □

The last lemma is a lower bound on the overlap, or inner product, of a quantum superposition of a discretized SDR function and an approximation to this state constructed with a fixed degree polynomial. This allows us to convert between the entrywise approximation error infinity-norm ℓ_∞ to a bound on the inner product or the two-norm ℓ_2.

Lemma 8.2.3. *Let f be a unit normalized discretization of a univariate SDR function $f(x)$ over a closed, bounded interval $D \subseteq \mathbb{R}$ covered by 2^N equidistant evaluation points, such that $\|f\|_2 = 1$. Let g be a unit vector pointwise approximation of f on a closed, bounded interval D such that $\|f - g\|_\infty = \varepsilon$. Then*

$$\langle f, g \rangle \geq 1 - \varepsilon^2 2^{N-1} \tag{8.7}$$

Proof. Begin by considering an ε-pointwise approximation g to f without requiring $\|g\|_2 = 1$. In this case, we can set:

$$g_i \equiv f_i - \varepsilon \quad \forall i \in [1, 2^N] \tag{8.8}$$

This choice for g minimizes $\langle f, g \rangle$ to:

$$\langle f, g \rangle = \sum_i^{2^N} f_i(f_i - \varepsilon) = \sum_i^{2^N} (f_i^2 - \varepsilon f_i) \tag{8.9}$$

$$= 1 - \varepsilon \sum_i^{2^N} f_i = 1 - \varepsilon \|f\|_1 \leq 1 - \varepsilon 2^{N/2} \tag{8.10}$$

Consider any other g' defined by changing some of the entries as:

$$g'_j = f_j + \varepsilon_j \tag{8.11}$$

for any $-\varepsilon < \varepsilon_j < \varepsilon$, with the only constraint that the entries follow $|f_j - g_j| \leq \varepsilon$ as dictated

by the pointwise approximation error. We then have:

$$\langle f, g' \rangle = \sum_{i}^{2^N} f_i(f_i + \varepsilon_i) = \sum_{i}^{2^N} (f_i^2 + \varepsilon_i f_i) \tag{8.12}$$

$$= 1 + \sum_{i}^{2^N} \varepsilon_i f_i \tag{8.13}$$

Minimizing equation 8.13 requires minimizing a summation over the distribution of ε_i, which is clearly minimized for $\varepsilon_i = -\varepsilon$ for all i.

Requiring that $\|g\|_2 = 1$ will tighten the bound:

$$\|g\|_2 = 1 = \sum_i g_i^2 = \sum_i (f_i + \varepsilon_i)^2 \tag{8.14}$$

$$= \sum_i (f_i^2 + 2\varepsilon_i f_i + \varepsilon_i^2) \tag{8.15}$$

$$= \sum_i f_i^2 + 2\sum_j \varepsilon_j f_j + \sum_k \varepsilon_k^2 \tag{8.16}$$

which implies by lemma 8.2.2 that:

$$-2\sum_i \varepsilon_i f_i = \sum_j \varepsilon_j^2 \leq 2^N \varepsilon^2 \tag{8.17}$$

$$\sum_i \varepsilon_i f_i \geq -2^{N-1}\varepsilon^2. \tag{8.18}$$

We can then tighten the bound in equation (8.10) to:

$$\langle f, g \rangle = \sum_i f_i g_i = \sum_i f_i(f_i + \varepsilon_i) \tag{8.19}$$

$$= \sum_i f_i^2 + \varepsilon_i f_i = 1 + \sum_i \varepsilon_i f_i \tag{8.20}$$

$$\geq 1 - 2^{N-1}\varepsilon^2 \tag{8.21}$$

□

Corollary 8.2.4. *For a fixed* error δ*, the required pointwise error* $\varepsilon = \|f - g\|_\infty$ *is given by:*

$$\varepsilon \leq \sqrt{\frac{\delta}{2^{N-1}}} \tag{8.22}$$

Proof. This is a simple rewriting of 8.7 with some fixed *error* δ so that:

$$\langle f, g \rangle \geq 1 - \varepsilon^2 2^{N-1} \geq (1 - \delta) \tag{8.23}$$

$$\varepsilon^2 \leq \frac{1 - (1 - \delta)}{2^{N-1}} \tag{8.24}$$

□

We now state the main result. By convention, we use the definition of the K-qubit reduced density matrix bipartite von-Neumann entropy as:

$$S(\rho_K) = -\mathrm{Tr}[\rho_K \log \rho_K] \tag{8.25}$$

$$= \sum_i |\lambda_i|^2 \log |\lambda_i|^2 \tag{8.26}$$

where ρ_K is a K-qubit reduced density matrix, and equation 8.26 holds for pure state Schmidt decompositions of ρ_K with λ_i Schmidt coefficients.

Theorem 8.2.5. *Let* ρ *be the quantum superposition corresponding to the discretization of a univariate SDR function* $f(x)$ *over domain* $D \subset \mathbb{R}$ *covered by* 2^N *equidistant evaluation points. Then:*

$$S_{max}(\rho_K) \leq \mathcal{O}(\log N) \tag{8.27}$$

where ρ_K *is any* K*-qubit reduced density matrix of* ρ*.*

This states that the maximum von-Neumann bipartite entropy of a discretized SDR

function as a quantum state grows as $\mathcal{O}(\log N)$. Unless explicitly specified otherwise, all logarithms are assumed to be base-2.

Proof. Begin by writing the $\varepsilon-$accurate pointwise approximation to $f(x)$ as g, such that $\|f-g\|_\infty = \varepsilon$. By Lemma 8.2.1, we state the degree p of a polynomial g required to achieve pointwise accuracy ε as:

$$p = \log_\alpha\left(1/\varepsilon\right) + C \tag{8.28}$$

for constant C and $\alpha = 1 + \frac{2}{\gamma_f |D|}$. By Lemma 8.2.3 and the corresponding Corollary 8.22, we write the required error ε^* for fixed error δ as:

$$\varepsilon^* \leq \sqrt{\frac{\delta}{2^{N-1}}} = \sqrt{\delta} 2^{-(N-1)/2} \tag{8.29}$$

Plugging equation 8.29 into equation 8.28 for constant error $\delta \ll 1$ we have:

$$p = \frac{\log\left(2^{(N-1)/2}/\sqrt{\delta}\right)}{\log\left(1+\frac{2}{\gamma_f|D|}\right)} \tag{8.30}$$

$$= \frac{\log\left(2^{(N-1)/2}\right) - \log\left(\sqrt{\delta}\right)}{\log(1+\frac{2}{\gamma_f|D|})} \tag{8.31}$$

$$= \frac{(N-1) - \log\delta}{2\log(1+\frac{2}{\gamma_f|D|})} \tag{8.32}$$

We now use the fact that a MPS encoding of a polynomial of degree p requires a maximum bond-dimension $\chi \leq p+1$ [89]. Because of this, and because the maximum achievable von-Neumann bipartite entropy of a rank-χ MPS is $\log_2(\chi)$ [66], we have, for constants

C_1, C_2:

$$S_{\mathrm{m}}(\rho_K) \leq \log(p+1) \tag{8.33}$$

$$= \log\left(\frac{(N-1) - \log\delta}{2\log(1+\frac{2}{\gamma_f|D|})} + 1\right) \tag{8.34}$$

$$= \log\left(\frac{N - 1 - \log\delta + 2\log(1+\frac{2}{\gamma_f|D|})}{2\log(1+\frac{2}{\gamma_f|D|})}\right) \tag{8.35}$$

$$= \log(N - \log\delta + C_1) - C_2 \tag{8.36}$$

$$= \mathcal{O}(\log N) \tag{8.37}$$

as stated. □

Theorem 8.2.5 indicates that SDR function discretizations as quantum superpositions have entanglement entropy that scales logarithmically with the size of the system. As others have shown [191], this implies efficient storage in a MPS data structure for all such functions.

Additionally, there is only a weak dependence on the maximum analytical derivative of the function: γ_f. As long as γ_f grows sub-exponentially, then the function will remain efficiently stored by a polynomial number of parameters in a MPS.

We can go farther and explore the accuracy of bounded bond-dimension (χ) MPS approximations to discretized SDR function superpositions. To do so, we invoke the Fannes-Audenaert entropy bound [12], and use it to upper-bound the overlap between an SDR function discretization superposition and a fixed-χ MPS approximation to these states. This yields an upper limit to the ability of compressed MPS forms to accurately approximate these states.

Corollary 8.2.6. *Let f be the unit normalized exact discretization of a univariate SDR function $f(x)$ over domain $D \subset \mathbb{R}$ covered by 2^N equidistant evaluation points, such that $\|f\|_2 = 1$. Also, let the corresponding quantum state ρ_f saturate the entropy bound in Theorem 8.2.5: $S_{max}(\rho_{f,K}) \leq \mathcal{O}(\log N) = C_0 \log N$ for some constant C_0. Let g be the unit normalized function corresponding to a rank-2 matrix product state approximation of f,*

denoted by σ_2. Then:

$$\|\rho - \sigma_2\|_{tr} = \Omega\left(\frac{\log N}{N}\right) \tag{8.38}$$

and for constants A, B:

$$\langle f, g\rangle = \mathcal{O}\left(\sqrt{1 - A^2\big(\log(N+B)/N\big)^2}\right) \tag{8.39}$$

Proof. We show this using the Fannes-Audenaert entropic bound [12].

$$|S(\rho) - S(\sigma_2)| \leq T\log(2^N - 1) + H((T, 1-T)) \tag{8.40}$$

$$\leq \frac{1}{2}\|\rho - \sigma_2\|_{\text{tr}}\log(2^N - 1) + 1 \tag{8.41}$$

where $T = \frac{1}{2}\|\rho - \sigma_2\|_{\text{tr}}$ and H is the binary Shannon entropy, upper bounded by 1.

By rearranging the bound, we see:

$$\|\rho - \sigma_2\|_{\text{tr}} \geq \frac{2|S(\rho) - S(\sigma_2)|}{\log(2^N - 1)} - 1 \tag{8.42}$$

$$= \frac{2}{N}\left|\frac{C_0\log(N - \log\delta + C_1)}{\log(C_2)} - 1\right| - 1 \tag{8.43}$$

$$= \frac{2C_0\log(N - \log\delta + C_1 - 2\log C_2)}{N\log(C_2)} \tag{8.44}$$

$$= \Omega\left(\frac{\log N}{N}\right) \tag{8.45}$$

This holds in the limit of large systems N, $\log(2^N - 1) \to N$, and with the maximum entropy saturating the upper bound provided by Theorem 8.2.5 and the maximum entropy of a rank-2 MPS as 1. $S(\rho) - S(\sigma_2)$ is assumed to be positive.

By exploiting the fact that these states are all pure states, we can convert the trace

distance to fidelity [153] and write:

$$\langle f, g\rangle = \sqrt{1 - \|f - g\|_{\text{tr}}^2} \tag{8.46}$$

$$\leq \sqrt{1 - \left(\frac{2\log(N - \log\delta + C_1 - 2\log C_2)}{N\log C_2}\right)^2} \tag{8.47}$$

$$= \mathcal{O}\left(\sqrt{1 - A^2\big(\log(N+B)/N\big)^2}\right) \tag{8.48}$$

□

Note that this is an upper bound on the trace distance between these two quantum states: the original, and a polynomial approximator. In the best cases, $S(\rho_{f,K})$ is bounded below $\log N$, in which case the approximator with constant entropy can competitively construct a high-accuracy state. These bounds imply that approximations can become exact for large system sizes.

8.3 Discussion

In conclusion, constructing quantum states that correspond to classical data is vital to preserving quantum speedup for applications that rely on large sets of classical input data. This work demonstrates that, as long as classical data sets are approximable by smooth, differentiable, real-valued functions, then the maximum bipartite von-Neumann entropy of the corresponding pure quantum state grows only logarithmically in the size of the system, or dataset. Equivalentally, the required entanglement necessary to fully describe the dataset scales only logarithmically with the amount of data being considered. As data becomes more unstructured, constant factors in equation (8.32) begin to increase the required polynomial approximation degree, which in turn requires more entanglement to fully describe the data inside a quantum register.

Corollary 8.2.6 shows that this logarithmic scaling actually indicates the existence of highly accurate, low-rank matrix product states that approximate these states, for large data

sets. Because the Fannes-Audenaert bound is tight, this upper bound on the state fidelity of a rank-2 approximation is good evidence that approaches like that of [102] will scale for larger data sets. In fact, Corollary 8.2.6 indicates that as long as the data is well-structured, the larger the data set becomes, the better the best rank-2 matrix product state approximation becomes. Based on this reasoning, the technique developed in [102] that seeks out this approximation is expected to scale efficiently and remain accurate for larger classical data sets and functions.

Part IV

Conclusion

CHAPTER 9

CONCLUSION

This dissertation has presented research across many of the integral components of quantum computing system design and construction. Quantum computing *architectural* research is quickly becoming a vital piece of innovation required to progress the construction of these systems beyond the small-scale. The engineering complexity involved in building larger and larger systems requires significant and immediate creative solutions in order to continue making progress. Architectural problems, from the fine-grained difficulties in wiring and hardware engineering, to the higher-level problems associated with architecting appropriately fast error-correcting mechanisms and decoder designs, all need to be solved in order for these machines to become a reality. We have presented sets of solutions that make progress along these lines, focusing on error-corrected machine architectures and decoder designs. Quantum computing algorithmic design and analysis is a field that also requires significant creative leaps in the near future in order to establish verifiable performance gains and prove the impacts that these machines will have. We have also presented research along these lines that aims to design and prove the efficiency of new state-preparation techniques and algorithmic protocols, and co-design algorithms with hardware. These efforts along with the architectural research have made progress towards bringing quantum computation into reality, and will hopefully serve as a foundation upon which others can continue to research and build.

Quantum computing remains one of the most ambitious scientific projects of the 21st century. As of today, vast amounts of public and private resources have been dedicated to the pursuit of this goal, and significant progress has been made over the past few decades. Those resources have accomplished much so far, and we can only anticipate continued success in the years to come.

Part V

Appendix

CHAPTER 10

PUBLICATION LIST

– Holmes, A., Matsuura, A. Y. (2020). Entanglement properties of quantum superpositions corresponding to smooth, differentiable functions. *Manuscript in preparation*

– Holmes, A., Matsuura, A. Y. (2020). Efficient quantum circuits for accurate state preparation of smooth, differentiable functions. *arXiv:2005.04351. To Appear: IEEE International Conference on Quantum Computing and Engineering (QCE20).*

– Holmes, A., Johri, S., Guerreschi, G. G., Clarke, J. S., Matsuura, A. Y. (2020). Impact of qubit connectivity on quantum algorithm performance. *Quantum Science and Technology.*

– Holmes A., Jokar, M. R., Pasandi, G., Ding, Y., Pedram, M., and Chong, F.t., (2020). NISQ+: Boosting computational power of quantum computers by approximating quantum error correction. *47th Annual International Symposium on Computer Architecture (ISCA).*

– Sawaya, Nicolas P. D., Guerreschi, G. G., Holmes, A., (2020). On connectivity-dependent resource requirements for digital quantum simulation of d-level particles. *arXiv:2005.13070. To Appear IEEE International Conference on Quantum Computing and Engineering (QCE20).*

– Ding, Y., Wu, X., Holmes, A., Wiseth, A., Franklin, D., Martonosi, M., and Chong, F.t., (2020). SQUARE: Strategic Quantum Ancilla Reuse for Modular Quantum Programs via Cost-Effective Uncomputation. *47th Annual International Symposium on Computer Architecture (ISCA).*

– Holmes, A., Ding, Y., Javadi-Abhari, A., Franklin, D., Martonosi, M. and Chong, F.T., (2019). Resource optimized quantum architectures for surface code implementations of magic-state distillation. *Microprocessors and Microsystems, 67, 56-70.*

– Ding, Y*., Holmes, A*., Javadi-Abhari, A., Franklin, D., Martonosi, M. and Chong, F., (2018), "Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures". *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) (pp. 828-840). IEEE.*

– Cui, W., Ding, Y., Dangwal, D., Holmes, A., McMahan, J., Javadi-Abhari, A., Tzimpragos, G., Chong, F. and Sherwood, T., (2018), June. Charm: a language for closed-form high-level architecture modeling. *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA) (pp. 152-165). IEEE.*

– Javadi-Abhari, A., Gokhale, P., Holmes, A., Franklin, D., Brown, K.R., Martonosi, M. and Chong, F.T., (2017), "Optimized surface code communication in superconducting quantum computers." *In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (pp. 692-705). ACM.*

– Javadi-Abhari, A., Holmes, A., Heckey, J., Patil, S., Kudrow, D., Franklin, D., Brown, K., Martonosi, M., Chong, F.T., (2017) "Compiler Management of Communication and Parallelism for Fault-Tolerant Quantum Computation." *ACM Transactions on Computing Systems.*

– Heckey, J., Patil, S., JavadiAbhari, A., Holmes, A., Kudrow, D., Brown, K.R., Franklin, D., Chong, F.T. and Martonosi, M., (2015), March. Compiler management of communication and parallelism for quantum computation. *ACM SIGARCH Computer Architecture News (Vol. 43, No. 1, pp. 445-456). ACM.*

– Eichhorn, R., Markham, S., Holmes, A., Sabol, D., Smith, E., (2013). Managing Parallel Cryogenic Flows to the Thermal Intercepts in the Cornell ERL. *AIP Publishing.*

– Eichhorn, R., Ganshin, A., Holmes, A., Kaufman, J., Markham, S., Posen, S., Smith, E., (2013). "Recent Findings on Nitrogen Treated Niobium". *SRF France.*

REFERENCES

[1] During the preparation of the manuscript, the authors became aware of a new implementation of the distillation procedure. While our analysis is based on an earlier implementation, the algorithm leverages the block code structure and should still be applicable.

[2] Figure holds for $n \geq 3$. depth becomes $5n - 4$ for $n < 3$.

[3] Ibm quantum experience. `https://quantumexperience.ng.bluemix.net/qx/devices`. Accessed: 2018-05-15.

[4] A preview of bristlecone, google's new quantum processor. `https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html`. Accessed: 2018-05-15.

[5] Results are presented for instantiations of the application that do not include gaps – i.e. all qubits are involved in the parity calculation.

[6] Reformulating chemistry for more efficient quantum computation, Mar 2018.

[7] Daniel S. Abrams and Seth Lloyd. Quantum algorithm providing exponential speed increase for finding eigenvalues and eigenvectors. *Physical Review Letters*, 83(24):5162–5165, dec 1999.

[8] Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time t-depth optimization of clifford+ t circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1476–1489, 2014.

[9] Hussain Anwar, Earl T Campbell, and Dan E Browne. Qutrit magic state distillation. *New Journal of Physics*, 14(6):063006, 2012.

[10] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[11] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[12] Koenraad MR Audenaert. A sharp continuity estimate for the von neumann entropy. *Journal of Physics A: Mathematical and Theoretical*, 40(28):8127, 2007.

[13] Ryan Babbush, Nathan Wiebe, Jarrod McClean, James McClain, Hartmut Neven, and Garnet Kin Chan. Low depth quantum simulation of electronic structure. *arXiv preprint arXiv:1706.00023*, 2017.

[14] Paul Baireuther, MD Caio, B Criger, Carlo WJ Beenakker, and Thomas E O'Brien. Neural network decoder for topological color codes with circuit level noise. *New Journal of Physics*, 21(1):013003, 2019.

[15] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical review A*, 52(5):3457, 1995.

[16] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457, 1995.

[17] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical review A*, 52(5):3457, 1995.

[18] Rami Barends, Julian Kelly, Anthony Megrant, Andrzej Veitia, Daniel Sank, Evan Jeffrey, Ted C White, Josh Mutus, Austin G Fowler, Brooks Campbell, et al. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508(7497):500, 2014.

[19] Rami Barends, Alireza Shabani, Lucas Lamata, Julian Kelly, Antonio Mezzacapo, Urtzi Las Heras, Ryan Babbush, Austin G Fowler, Brooks Campbell, Yu Chen, et al. Digitized adiabatic quantum computing with a superconducting circuit. *Nature*, 534(7606):222–226, 2016.

[20] Earl R Barnes. An algorithm for partitioning the nodes of a graph. *SIAM Journal on Algebraic Discrete Methods*, 3(4):541–550, 1982.

[21] Angelo Bassi and Dirk-André Deckert. Noise gates for decoherent quantum circuits. *Physical Review A*, 77(3):032323, 2008.

[22] CD Batista and Gerardo Ortiz. Generalized jordan-wigner transformations. *Physical review letters*, 86(6):1082, 2001.

[23] Kim Batselier, Wenjian Yu, Luca Daniel, and Ngai Wong. Computing low-rank approximations of large-scale matrices with the tensor network randomized svd. *SIAM Journal on Matrix Analysis and Applications*, 39(3):1221–1244, 2018.

[24] Jacob Biamonte and Ville Bergholm. Tensor networks in a nutshell. *arXiv preprint arXiv:1708.00006*, 2017.

[25] Lev S Bishop, Sergey Bravyi, Andrew Cross, Jay M Gambetta, and John Smolin. Quantum volume. Technical report, Technical report, 2017. URL: https://dal. objectstorage. open. softlayer. com/v1/AUTH_039c3bf6e6e54d76b8e66152e2f87877/communitydocuments/quatnum-volumehp08co1vbo0cc8fr. pdf, 2017.

[26] Felix Bloch. Nuclear induction. *Physical review*, 70(7-8):460, 1946.

[27] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[28] Héctor Bombín. Structure of 2d topological stabilizer codes. *Communications in Mathematical Physics*, 327(2):387–432, 2014.

[29] Steffen Börm, Maike Löhndorf, and Jens M Melenk. Approximation of integral operators by variable-order interpolation. *Numerische Mathematik*, 99(4):605–643, 2005.

[30] Katherine Bourzac. Chemistry is quantum computing's killer app. *Chemical Engineering News*, 95(43):27–31, 2017.

[31] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Quantum Computation and Information, Contemporary Mathematics*, 305:53–74, 2002.

[32] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[33] Sergey Bravyi and David Gosset. Improved classical simulation of quantum circuits dominated by clifford gates. *Physical review letters*, 116(25):250501, 2016.

[34] Sergey Bravyi and Jeongwan Haah. Magic-state distillation with low overhead. *Phys. Rev. A*, 86:052329, Nov 2012.

[35] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A*, 71(2), 2005.

[36] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A*, 71(2):022316, 2005.

[37] Sergey Bravyi, Martin Suchara, and Alexander Vargo. Efficient algorithms for maximum likelihood decoding in the surface code. *Physical Review A*, 90(3):032326, 2014.

[38] Stephen Brierley. Efficient implementation of Quantum circuits with limited qubit interactions. *arXiv:1507.04263v2*, 2015.

[39] Kenneth R Brown, Jungsang Kim, and Christopher Monroe. Co-designing a scalable quantum computer with trapped atomic ions. *npj Quantum Information*, 2:16034, 2016.

[40] Peter Bürgisser, Michael Clausen, and Mohammad A Shokrollahi. *Algebraic complexity theory*, volume 315. Springer Science & Business Media, 2013.

[41] Earl T Campbell, Hussain Anwar, and Dan E Browne. Magic-state distillation in all prime dimensions using quantum reed-muller codes. *Physical Review X*, 2(4):041021, 2012.

[42] Christopher Chamberland and Pooya Ronagh. Deep neural decoders for near term fault-tolerant experiments. *Quantum Science and Technology*, 3(4):044002, 2018.

[43] Maolin Che and Yimin Wei. Randomized algorithms for the approximations of tucker and the tensor train decompositions. *Advances in Computational Mathematics*, 45(1):395–428, 2019.

[44] W Chen, AV Rylyakov, Vijay Patel, JE Lukens, and KK Likharev. Rapid single flux quantum t-flip flop operating up to 770 ghz. *IEEE Transactions on Applied Superconductivity*, 9(2):3212–3215, 1999.

[45] Andrew M Childs, Robin Kothari, and Rolando D Somma. Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. *SIAM Journal on Computing*, 46(6):1920–1950, 2017.

[46] Jerry M Chow, Jay M Gambetta, AD Córcoles, Seth T Merkel, John A Smolin, Chad Rigetti, S Poletto, George A Keefe, Mary B Rothwell, JR Rozen, et al. Universal quantum gate set approaching fault-tolerant thresholds with superconducting qubits. *Physical review letters*, 109(6):060501, 2012.

[47] Julia Chuzhoy, Yury Makarychev, and Anastasios Sidiropoulos. On graph crossing number and edge planarization. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete algorithms*, pages 1050–1069. SIAM, 2011.

[48] Patrick J Coles, Stephan Eidenbenz, Scott Pakin, Adetokunbo Adedoyin, John Ambrosiano, Petr Anisimov, William Casper, Gopinath Chennupati, Carleton Coffrin, Hristo Djidjev, et al. Quantum algorithm implementations for beginners. *arXiv preprint arXiv:1804.03719*, 2018.

[49] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536:63, 2016.

[50] Nicolas Delfosse and Naomi H Nickerson. Almost-linear time decoding algorithm for topological codes. *arXiv preprint arXiv:1709.06218*, 2017.

[51] Nicolas Delfosse and Gilles Zémor. Linear-time maximum likelihood decoding of surface codes over the quantum erasure channel. *arXiv preprint arXiv:1703.01517*, 2017.

[52] Johannes Arnoldus Delport, Kyle Jackman, Paul Le Roux, and Coenrad Johann Fourie. Josim - superconductor spice simulator. *IEEE Transactions on Applied Superconductivity*, 29(5):1–5, 2019.

[53] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[54] Yongshan Ding, Adam Holmes, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic Chong. Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 828–840. IEEE, 2018.

[55] Sergey Dolgov, Karim Anaya-Izquierdo, Colin Fox, and Robert Scheichl. Approximation and sampling of multivariate probability distributions in the tensor train decomposition. *Statistics and Computing*, 30(3):603–625, 2020.

[56] Sergey V Dolgov, Boris N Khoromskij, Ivan V Oseledets, and Dmitry V Savostyanov. Computation of extreme eigenvalues in higher dimensions using block tensor train format. *Computer Physics Communications*, 185(4):1207–1216, 2014.

[57] William E Donath and Alan J Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973.

[58] Kavita Dorai and Dieter Suter. Efficient implementations of the quantum fourier transform: an experimental perspective. *International Journal of Quantum Information*, 3(02):413–424, 2005.

[59] Doratha E Drake and Stefan Hougardy. A simple approximation algorithm for the weighted matching problem. *Information Processing Letters*, 85(4):211–213, 2003.

[60] Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104, 2005.

[61] Guillaume Duclos-Cianci and David Poulin. Fast decoders for topological quantum codes. *Physical review letters*, 104(5):050504, 2010.

[62] Guillaume Duclos-Cianci and David Poulin. A renormalization group decoding algorithm for topological quantum codes. In *2010 IEEE Information Theory Workshop*, pages 1–5. IEEE, 2010.

[63] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[64] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[66] Jens Eisert. Entanglement and tensor network states. *arXiv preprint arXiv:1308.3318*, 2013.

[67] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

[68] Caroline Figgatt, Aaron Ostrander, Norbert M Linke, Kevin A Landsman, Daiwei Zhu, Dmitri Maslov, and Christopher Monroe. Parallel entangling operations on a universal ion-trap quantum computer. *Nature*, 572(7769):368–372, 2019.

[69] Austin G. Fowler. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3), 2012.

[70] Austin G Fowler. Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $o(1)$ parallel time. *arXiv preprint arXiv:1307.1740*, 2013.

[71] 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, 2004.

[72] Austin G Fowler, Simon J Devitt, and Cody Jones. Surface code implementation of block code state distillation. *Scientific reports*, 3, 2013.

[73] Austin G Fowler, Simon J Devitt, and Cody Jones. Surface code implementation of block code state distillation. *Scientific Reports*, 3:1939, jun 2013.

[74] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[75] Austin G Fowler, Adam C Whiteside, and Lloyd CL Hollenberg. Towards practical classical processing for the surface code. *Physical review letters*, 108(18):180501, 2012.

[76] Austin G Fowler, Adam C Whiteside, and Lloyd CL Hollenberg. Towards practical classical processing for the surface code: timing analysis. *Physical Review A*, 86(4):042313, 2012.

[77] Austin G Fowler, Adam C Whiteside, Angus L McInnes, and Alimohammad Rabbani. Topological code autotune. *Physical Review X*, 2(4):041003, 2012.

[78] David P Franke, James S Clarke, Lieven MK Vandersypen, and Menno Veldhorst. Rent's rule and extensibility in quantum computing. *arXiv preprint arXiv:1806.02145*, 2018.

[79] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[80] Xiang Fu, MA Rol, CC Bultink, J van Someren, Nader Khammassi, Imran Ashraf, RFL Vermeulen, JC de Sterke, WJ Vlothuizen, RN Schouten, et al. A microarchitecture for a superconducting quantum processor. *IEEE Micro*, 38(3):40–47, 2018.

[81] Xiang Fu, Michiel Adriaan Rol, Cornelis Christiaan Bultink, J Van Someren, Nader Khammassi, Imran Ashraf, RFL Vermeulen, JC De Sterke, WJ Vlothuizen, RN Schouten, et al. An experimental microarchitecture for a superconducting quantum processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 813–825. ACM, 2017.

[82] Juan José García-Ripoll. Quantum-inspired algorithms for multivariate analysis: from interpolation to partial differential equations. *arXiv preprint arXiv:1909.06619*, 2019.

[83] Michael R Garey and David S Johnson. Crossing number is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[84] Joydip Ghosh, Austin G Fowler, and Michael R Geller. Surface code with decoherence: An analysis of three superconducting architectures. *Physical Review A*, 86(6):062318, 2012.

[85] G. Giacomo Guerreschi and J. Park. Gate scheduling for quantum algorithms. *ArXiv e-prints*, July 2017.

[86] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[87] GH Golub and CF Van Loan. Matrix computations 4th edition the johns hopkins university press. *Baltimore, MD*, 2013.

[88] Daniel Gottesman. Stabilizer codes and quantum error correction. *arXiv preprint quant-ph/9705052*, 1997.

[89] Lars Grasedyck. *Polynomial approximation in hierarchical Tucker format by vector-tensorization*. Inst. für Geometrie und Praktische Mathematik, 2010.

[90] Lov Grover and Terry Rudolph. Creating superpositions that correspond to efficiently integrable probability distributions. *arXiv preprint quant-ph/0208112*, 2002.

[91] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79(2):325–328, 1997.

[92] Gian Giacomo Guerreschi and Jongsoo Park. Gate scheduling for quantum algorithms. *arXiv preprint arXiv:1708.00023*, 2017.

[93] Jeongwan Haah, Matthew B Hastings, D Poulin, and D Wecker. Magic state distillation with low space overhead and optimal asymptotic input count. *arXiv preprint arXiv:1703.07847*, 2017.

[94] Hartmut Häffner, Wolfgang Hänsel, CF Roos, Jan Benhelm, Michael Chwalla, Timo Körber, UD Rapol, Mark Riebe, PO Schmidt, Christoph Becher, et al. Scalable multiparticle entanglement of trapped ions. *Nature*, 438(7068):643, 2005.

[95] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.

[96] Matthew B Hastings, Dave Wecker, Bela Bauer, and Matthias Troyer. Improving quantum algorithms for quantum chemistry. *arXiv preprint arXiv:1403.1539*, 2014.

[97] Quentin P Herr, Anna Y Herr, Oliver T Oberg, and Alexander G Ioannidis. Ultra-low-power superconductor logic. *Journal of applied physics*, 109(10):103903, 2011.

[98] Yuichi Hirata, Masaki Nakanishi, Shigeru Yamashita, and Yasuhiko Nakashima. An efficient method to convert arbitrary quantum circuits to ones on a linear nearest neighbor architecture. In *Quantum, Nano and Micro Technologies, 2009. ICQNM'09. Third International Conference on*, pages 26–33. IEEE, 2009.

[99] Adam Holmes, Yongshan Ding, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic T Chong. Resource optimized quantum architectures for surface code implementations of magic-state distillation. *Microprocessors and Microsystems*, 67:56–70, 2019.

[100] Adam Holmes, Sonika Johri, Gian Giacomo Guerreschi, James S Clarke, and AY Matsuura. Impact of qubit connectivity on quantum algorithm performance. *arXiv preprint arXiv:1811.02125*, 2018.

[101] Adam Holmes, Mohammad Reza Jokar, Ghasem Pasandi, Yongshan Ding, Massoud Pedram, and Frederic T Chong. Nisq+: Boosting quantum computing power by approximating quantum error correction. *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020.

[102] Adam Holmes and AY Matsuura. Efficient quantum circuits for accurate state preparation of smooth, differentiable functions. *IEEE International Conference on Quantum Computing Engineering (QCE20)*
arXiv preprint arXiv:2005.04351, 2020.

[103] Adam Holmes and AY Matsuura. Entanglement properties of quantum superpositions of smooth, differentiable functions. *arXiv preprint arXiv:2009.09096*, 2020.

[104] Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. The alternating linear scheme for tensor optimization in the tensor train format. *SIAM Journal on Scientific Computing*, 34(2):A683–A713, 2012.

[105] JM Hornibrook, JI Colless, ID Conway Lamb, SJ Pauka, H Lu, AC Gossard, JD Watson, GC Gardner, S Fallahi, MJ Manfra, et al. Cryogenic control architecture for large-scale quantum computing. *Physical Review Applied*, 3(2):024010, 2015.

[106] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[107] Hsieh Hou and H Andrews. Cubic splines for image interpolation and digital filtering. *IEEE Transactions on acoustics, speech, and signal processing*, 26(6):508–517, 1978.

[108] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[109] Benjamin Huber, Reinhold Schneider, and Sebastian Wolf. A randomized tensor train singular value decomposition. In *Compressed Sensing and its Applications*, pages 261–290. Springer, 2017.

[110] Barry D Hughes. Random walks and random environments. 1995.

[111] Nemanja Isailovic, Mark Whitney, Yatish Patel, and John Kubiatowicz. Running a

quantum circuit at the speed of data. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 177–188. IEEE Computer Society, 2008.

[112] Ali Javadi-Abhari, Pranav Gokhale, Adam Holmes, Diana Franklin, Kenneth R Brown, Margaret Martonosi, and Frederic T Chong. Optimized surface code communication in superconducting quantum computers. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 692–705. ACM, 2017.

[113] Ali JavadiAbhari et al. Scaffold: Quantum programming language. Technical report, Princeton University, 2012.

[114] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. Scaffcc: a framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, page 1. ACM, 2014.

[115] Sonika Johri, Damian S Steiger, and Matthias Troyer. Entanglement spectroscopy on a quantum computer. *Physical Review B*, 96(19):195136, 2017.

[116] Cody Jones. Multilevel distillation of magic states for quantum computing. *Physical Review A*, 87(4):042305, 2013.

[117] N Cody Jones, Rodney Van Meter, Austin G Fowler, Peter L McMahon, Jungsang Kim, Thaddeus D Ladd, and Yoshihisa Yamamoto. Layered architecture for quantum computing. *Physical Review X*, 2(3):031007, 2012.

[118] N Cody Jones, Rodney Van Meter, Austin G Fowler, Peter L McMahon, Jungsang Kim, Thaddeus D Ladd, and Yoshihisa Yamamoto. Layered architecture for quantum computing. *Physical Review X*, 2(3):031007, 2012.

[119] N Cody Jones, James D Whitfield, Peter L McMahon, Man-Hong Yung, Rodney Van Meter, Alán Aspuru-Guzik, and Yoshihisa Yamamoto. Faster quantum chemistry

simulation on fault-tolerant quantum computers. *New Journal of Physics*, 14(11):115023, 2012.

[120] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892, 2002.

[121] George Karypis and Vipin Kumar. Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995.

[122] George Karypis and Vipin Kumar. Multilevel k-way hypergraph partitioning. *VLSI design*, 11(3):285–300, 2000.

[123] Phillip Kaye and Michele Mosca. Quantum networks for generating arbitrary quantum states. *arXiv preprint quant-ph/0407102*, 2004.

[124] J. Kelly, Rami Barends, Austin G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Yu Chen, Z. Chen, B. Chiaro, A. Dunsworth, I.-C. Hoi, Charles Neill, P. J. J. O'Malley, C. Quintana, Pedran Roushan, A. Vainsencher, J. Wenner, A. N. Cleland, and John M. Martinis. State preservation by repetitive error detection in a superconducting quantum circuit. *Nature*, 519(7541):66–69, 2015.

[125] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[126] Robert Keys. Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160, 1981.

[127] Mozammel HA Khan. Cost reduction in nearest neighbour based synthesis of quantum boolean circuits. *Engineering Letters*, 16(1), 2008.

[128] DE Kirichenko, Saad Sarwana, and AF Kirichenko. Zero static power dissipation biasing of rsfq circuits. *IEEE Transactions on Applied Superconductivity*, 21(3):776–779, 2011.

[129] Alexei Kitaev and William A Webb. Wavefunction preparation and resampling using a quantum computer. *arXiv preprint arXiv:0801.0342*, 2008.

[130] Ian D Kivlichan, Jarrod McClean, Nathan Wiebe, Craig Gidney, Alán Aspuru-Guzik, Garnet Kin-Lic Chan, and Ryan Babbush. Quantum simulation of electronic structure with linear depth and connectivity. *Physical review letters*, 120(11):110501, 2018.

[131] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single qubit unitaries generated by clifford and t gates. *arXiv preprint arXiv:1206.5236*, 2012.

[132] Namgil Lee and Andrzej Cichocki. Very large-scale singular value decomposition using tensor train networks. *arXiv preprint arXiv:1410.6895*, 2014.

[133] Namgil Lee and Andrzej Cichocki. Estimating a few extreme singular values and vectors for large-scale matrices in tensor train format. *SIAM Journal on Matrix Analysis and Applications*, 36(3):994–1014, 2015.

[134] Seungyong Lee, George Wolberg, and Sung Yong Shin. Scattered data interpolation with multilevel b-splines. *IEEE transactions on visualization and computer graphics*, 3(3):228–244, 1997.

[135] Bjoern Lekitsch, Sebastian Weidt, Austin G Fowler, Klaus Mølmer, Simon J Devitt, Christof Wunderlich, and Winfried K Hensinger. Blueprint for a microwave trapped ion quantum computer. *Science Advances*, 3(2):e1601540, 2017.

[136] Gavriela Freund Lev, Nicholas Pippenger, and Leslie G Valiant. A fast parallel algorithm

for routing in permutation networks. *IEEE transactions on Computers*, 100(2):93–100, 1981.

[137] James E Levy, Malcolm S Carroll, Anand Ganti, Cynthia A Phillips, Andrew J Landahl, Thomas M Gurrieri, Robert D Carr, Harold L Stalford, and Erik Nielsen. Implications of electronics constraints for solid-state quantum error correction and quantum circuit failure probability. *New Journal of Physics*, 13(8):083021, 2011.

[138] James E Levy, Anand Ganti, Cynthia A Phillips, Benjamin R Hamlet, Andrew J Landahl, Thomas M Gurrieri, Robert D Carr, and Malcolm S Carroll. The impact of classical electronics constraints on a solid-state logical qubit memory. *arXiv preprint arXiv:0904.0003*, 2009.

[139] R Li, L Petit, DP Franke, JP Dehollain, J Helsen, M Steudtner, NK Thomas, ZR Yoscovits, KJ Singh, S Wehner, et al. A crossbar network for silicon quantum dot qubits. *arXiv preprint arXiv:1711.03807*, 2017.

[140] Daniel A Lidar and Todd A Brun. *Quantum error correction*. Cambridge university press, 2013.

[141] Konstantin K Likharev and Vasilii K Semenov. Rsfq logic/memory family: A new josephson-junction technology for sub-terahertz-clock-frequency digital systems. *IEEE Transactions on Applied Superconductivity*, 1(1):3–28, 1991.

[142] Chun-Cheng Lin and Hsu-Chun Yen. A new force-directed graph drawing method based on edge–edge repulsion. *Journal of Visual Languages & Computing*, 23(1):29–42, 2012.

[143] Norbert M Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A Landsman, Kenneth Wright, and Christopher Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305–3310, 2017.

[144] Daniel Litinski and Felix von Oppen. Lattice surgery with a twist: simplifying clifford gates of surface codes. *Quantum*, 2:62, 2018.

[145] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, 2014.

[146] Michael Lubasch, Jaewoo Joo, Pierre Moinier, Martin Kiffner, and Dieter Jaksch. Variational quantum algorithms for nonlinear problems. *Physical Review A*, 101(1):010301, 2020.

[147] Michael Lubasch, Pierre Moinier, and Dieter Jaksch. Multigrid renormalization. *Journal of Computational Physics*, 372:587–602, 2018.

[148] Ana Martin, Bruno Candelas, Ángel Rodríguez-Rozas, José D Martín-Guerrero, Xi Chen, Lucas Lamata, Román Orús, Enrique Solano, and Mikel Sanz. Towards pricing financial derivatives with an ibm quantum computer. *arXiv preprint arXiv:1904.05803*, 2019.

[149] Dmitri Maslov. Basic circuit compilation techniques for an ion-trap quantum machine. *New Journal of Physics*, 19(2):023035, 2017.

[150] Adam M Meier, Bryan Eastin, and Emanuel Knill. Magic-state distillation with the four-qubit code. *arXiv preprint arXiv:1204.4221*, 2012.

[151] Ashley Montanaro. Quantum speedup of monte carlo methods. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2181):20150301, 2015.

[152] Ashley Montanaro. Quantum algorithms: an overview. *npj Quantum Information*, 2:npjqi201523, 2016.

[153] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[154] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[155] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition, 2011.

[156] Joe O'Gorman and Earl T. Campbell. Quantum computation with realistic magic-state factories. *Phys. Rev. A*, 95:032338, Mar 2017.

[157] G. Ortiz, J. Gubernatis, Emanuel Knill, and Raymond Laflamme. Quantum algorithms for fermionic simulations. *Physical Review A*, 64(2):022319, jul 2001.

[158] Roman Orus, Samuel Mugel, and Enrique Lizaso. Quantum computing for finance: overview and prospects. *Reviews in Physics*, page 100028, 2019.

[159] IV Oseledets. Constructive representation of functions in low-rank tensor formats. *Constructive Approximation*, 37(1):1–18, 2013.

[160] Ivan Oseledets and Eugene Tyrtyshnikov. Tt-cross approximation for multidimensional arrays. *Linear Algebra and its Applications*, 432(1):70–88, 2010.

[161] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[162] Adam Paetznick and Ben W Reichardt. Fault-tolerant ancilla preparation and noise threshold lower bounds for the 23-qubit golay code. *arXiv preprint arXiv:1106.2190*, 2011.

[163] Alexandru Paler, Simon Devitt, Kae Nemoto, and Ilia Polian. Synthesis of topological quantum circuits. In *Proceedings of the 2012 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 181–187. ACM, 2012.

[164] Alexandru Paler, Simon J Devitt, and Austin G Fowler. Synthesis of arbitrary quantum circuits to topological assembly. *arXiv preprint arXiv:1604.08621*, 2016.

[165] Alexandru Paler, Ilia Polian, Kae Nemoto, and Simon J Devitt. A compiler for fault-tolerant high level quantum circuits. *arXiv preprint arXiv:1509.02004*, 2015.

[166] Alexandru Paler, Ilia Polian, Kae Nemoto, and Simon J Devitt. Fault-tolerant, high-level quantum circuits: form, compilation and description. *Quantum Science and Technology*, 2(2):025003, 2017.

[167] GS Paraoanu. Microwave-induced coupling of superconducting qubits. *Physical Review B*, 74(14):140504, 2006.

[168] J Anthony Parker, Robert V Kenyon, and Donald E Troxel. Comparison of interpolating methods for image resampling. *IEEE Transactions on medical imaging*, 2(1):31–39, 1983.

[169] Ghasem Pasandi and Massoud Pedram. Balanced factorization and rewriting algorithms for synthesizing single flux quantum logic circuits. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI (GLSVLSI)*, pages 183–188, 2019.

[170] Ghasem Pasandi and Massoud Pedram. A dynamic programming-based path balancing technology mapping algorithm targeting area minimization. In *Proc. IEEE/ACM Int. Conf. Comput. Aided Des. (ICCAD)*, 2019.

[171] Ghasem Pasandi and Massoud Pedram. PBMap: A path balancing technology mapping algorithm for single flux quantum logic circuits. *IEEE Transactions on Applied Superconductivity*, 29(4):1–14, 2019.

[172] Ghasem Pasandi, Alireza Shafaei, and Massoud Pedram. SFQmap: A technology mapping tool for single flux quantum logic circuits. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[173] Bishnu Patra, Rosario M Incandela, Jeroen PG Van Dijk, Harald AR Homulle, Lin Song, Mina Shahmohammadi, Robert Bogdan Staszewski, Andrei Vladimirescu, Masoud Babaie, Fabio Sebastiano, et al. Cryo-cmos circuits and systems for quantum computing applications. *IEEE Journal of Solid-State Circuits*, 53(1):309–321, 2018.

[174] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *International Conference on High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[175] David Perez-Garcia, Frank Verstraete, Michael M Wolf, and J Ignacio Cirac. Matrix product state representations. *arXiv preprint quant-ph/0608197*, 2006.

[176] JH Plantenberg, PC De Groot, CJPM Harmans, and JE Mooij. Demonstration of controlled-not quantum gates on a pair of superconducting quantum bits. *Nature*, 447(7146):836, 2007.

[177] Martin Plesch and Časlav Brukner. Quantum-state preparation with universal gate decompositions. *Physical Review A*, 83(3):032302, 2011.

[178] John Preskill. Quantum computing in the nisq era and beyond. *arXiv preprint arXiv:1801.00862*, 2018.

[179] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[180] Sergi Ramos-Calderer, Adrián Pérez-Salinas, Diego García-Martín, Carlos Bravo-Prieto, Jorge Cortada, Jordi Planagumà, and José I Latorre. Quantum unary approach to option pricing. *arXiv preprint arXiv:1912.01618*, 2019.

[181] Shi-Ju Ran. Encoding of matrix product states into quantum circuits of one-and two-qubit gates. *Physical Review A*, 101(3):032310, 2020.

[182] Patrick Rebentrost, Brajesh Gupt, and Thomas R Bromley. Quantum computational finance: Monte carlo pricing of financial derivatives. *Physical Review A*, 98(2):022321, 2018.

[183] Patrick Rebentrost and Seth Lloyd. Quantum computational finance: quantum algorithm for portfolio optimization. *arXiv preprint arXiv:1811.03975*, 2018.

[184] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503, 2014.

[185] David Rosenbaum. Optimal Quantum Circuits for Nearest-Neighbor Architectures. *arXiv:1205.0036*, 2012.

[186] Neil J Ross and Peter Selinger. Optimal ancilla-free clifford+ t approximation of z-rotations. *arXiv preprint arXiv:1403.2975*, 2014.

[187] Mehdi Saeedi, Robert Wille, and Rolf Drechsler. Synthesis of quantum circuits for linear nearest neighbor architectures. *Quantum Information Processing*, 10(3):355–377, 2011.

[188] Nicolas PD Sawaya, Mikhail Smelyanskiy, Jarrod R McClean, and Alán Aspuru-Guzik. Error sensitivity to environmental noise in quantum circuits for chemical state preparation. *Journal of chemical theory and computation*, 12(7):3097–3108, 2016.

[189] Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 138–148. Society for Industrial and Applied Mathematics, 1990.

[190] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192, 2011.

[191] Norbert Schuch, Michael M Wolf, Frank Verstraete, and J Ignacio Cirac. Entropy

scaling and simulability by matrix product states. *Physical review letters*, 100(3):030504, 2008.

[192] T Schulte-Herbrüggen, A Spörl, and SJ Glaser. Quantum cisc compilation by optimal control and scalable assembly of complex instruction sets beyond two-qubit gates. *arXiv preprint arXiv:0712.3227*, 2007.

[193] Fabio Sebastiano, Harald Homulle, Bishnu Patra, Rosario Incandela, Jeroen van Dijk, Lin Song, Masoud Babaie, Andrei Vladimirescu, and Edoardo Charbon. Cryo-cmos electronic control for scalable quantum computing. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 13. ACM, 2017.

[194] Peter Selinger. Quantum circuits of t-depth one. *Physical Review A*, 87(4), 2013.

[195] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In *Proceedings of the 50th Annual Design Automation Conference*, page 41. ACM, 2013.

[196] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. Qubit placement to minimize communication overhead in 2d quantum architectures. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 495–500. IEEE, 2014.

[197] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[198] Mikhail Smelyanskiy, Nicolas PD Sawaya, and Alán Aspuru-Guzik. qhipster: The quantum high performance software testing environment. *arXiv preprint arXiv:1601.07195*, 2016.

[199] Nikitas Stamatopoulos, Daniel J Egger, Yue Sun, Christa Zoufal, Raban Iten, Ning Shen, and Stefan Woerner. Option pricing using quantum computers. *arXiv preprint arXiv:1905.02666*, 2019.

[200] Andrew Steane. Space, time, parallelism and noise requirements for reliable quantum computing. *arXiv preprint quant-ph/9708021*, 1997.

[201] Krysta M Svore and Matthias Troyer. The quantum future of computation. *Computer*, 49(9):21–30, 2016.

[202] Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. Quantum addition circuits and unbounded fan-out. *arXiv preprint arXiv:0910.2530*, 2009.

[203] Naoki Takeuchi, Dan Ozawa, Yuki Yamanashi, and Nobuyuki Yoshikawa. An adiabatic quantum flux parametron as an ultra-low-power logic device. *Superconductor Science and Technology*, 26(3):035010, 2013.

[204] Swamit S Tannu, Douglas M Carmean, and Moinuddin K Qureshi. Cryogenic-dram based memory system for scalable quantum computers: a feasibility study. In *Proceedings of the International Symposium on Memory Systems*, pages 189–195. ACM, 2017.

[205] Swamit S Tannu, Zachary A Myers, Prashant J Nair, Douglas M Carmean, and Moinuddin K Qureshi. Taming the instruction bandwidth of quantum computers via hardware-managed error correction. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 679–691. ACM, 2017.

[206] Barbara M Terhal. Quantum error correction for quantum memories. *Reviews of Modern Physics*, 87(2):307, 2015.

[207] Yu Tomita and Krysta M Svore. Low-distance surface codes under realistic quantum noise. *Physical Review A*, 90(6):062320, 2014.

[208] Giacomo Torlai and Roger G Melko. Neural decoder for topological codes. *Physical review letters*, 119(3):030501, 2017.

[209] William F Trench. Elementary differential equations with boundary value problems. 2013.

[210] Hale F Trotter. On the product of semi-groups of operators. *Proceedings of the American Mathematical Society*, 10(4):545–551, 1959.

[211] Rodney Van Meter and Clare Horsman. A blueprint for building a quantum computer. *Communications of the ACM*, 56(10):84–93, 2013.

[212] Savvas Varsamopoulos, Koen Bertels, and Carmen G Almudever. Designing neural network based decoders for surface codes. *arXiv preprint arXiv:1811.12456*, 2018.

[213] Savvas Varsamopoulos, Koen Bertels, and Carmen G Almudever. Designing neural network based decoders for surface codes. *arXiv preprint arXiv:1811.12456*, 2018.

[214] Savvas Varsamopoulos, Koen Bertels, and Carmen G Almudever. Decoding surface code with a distributed neural network based decoder. *arXiv preprint arXiv:1901.10847*, 2019.

[215] Savvas Varsamopoulos, Koen Bertels, and Carmen Garcia Almudever. Comparing neural network based decoders for the surface code. *IEEE Transactions on Computers*, 2019.

[216] Savvas Varsamopoulos, Ben Criger, and Koen Bertels. Decoding small surface codes with feedforward neural networks. *Quantum Science and Technology*, 3(1):015004, 2017.

[217] Savvas Varsamopoulos, Ben Criger, and Koen Bertels. Decoding small surface codes with feedforward neural networks. *Quantum Science and Technology*, 3(1):015004, 2017.

[218] R Versluis, S Poletto, N Khammassi, B Tarasinski, N Haider, DJ Michalak, A Bruno, K Bertels, and L DiCarlo. Scalable quantum circuit and control for a superconducting surface code. *Physical Review Applied*, 8(3):034021, 2017.

[219] Guifré Vidal. Efficient classical simulation of slightly entangled quantum computations. *Physical review letters*, 91(14):147902, 2003.

[220] Guifré Vidal. Efficient simulation of one-dimensional quantum many-body systems. *Physical review letters*, 93(4):040502, 2004.

[221] Mark H Volkmann, Anubhav Sahu, Coenrad J Fourie, and Oleg A Mukhanov. Experimental investigation of energy-efficient digital circuits based on esfq logic. *IEEE Transactions on Applied Superconductivity*, 23(3):1301505–1301505, 2013.

[222] Fred Ware, Liji Gopalakrishnan, Eric Linstadt, Sally A McKee, Thomas Vogelsang, Kenneth L Wright, Craig Hampel, and Gary Bronner. Do superconducting processors really need cryogenic memories?: the case for cold dram. In *Proceedings of the International Symposium on Memory Systems*, pages 183–188. ACM, 2017.

[223] Dave Wecker, Bela Bauer, Bryan K Clark, Matthew B Hastings, and Matthias Troyer. Gate-count estimates for performing quantum chemistry on small quantum computers. *Physical Review A*, 90(2):022305, 2014.

[224] James D Whitfield, Jacob Biamonte, and Alán Aspuru-Guzik. Simulation of electronic structure hamiltonians using quantum computers. *Molecular Physics*, 109(5):735–750, 2011.

[225] James D Whitfield, Jacob Biamonte, and Alán Aspuru-Guzik. Simulation of electronic structure hamiltonians using quantum computers. *Molecular Physics*, 109(5):735–750, 2011.

[226] Nathan Wiebe, Daniel Braun, and Seth Lloyd. Quantum algorithm for data fitting. *Physical review letters*, 109(5):050505, 2012.

[227] Robert Wille, Aaron Lye, and Rolf Drechsler. Optimal swap gate insertion for nearest neighbor quantum circuits. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 489–494. IEEE, 2014.

[228] Stefan Woerner and Daniel J Egger. Quantum risk analysis. *npj Quantum Information*, 5(1):1–8, 2019.

[229] James Wootton. A simple decoder for topological codes. *Entropy*, 17(4):1946–1957, 2015.

[230] Chui-Ping Yang, Shih-I Chu, and Siyuan Han. Possible realization of entanglement, logical gates, and quantum-information transfer with superconducting-quantum-interference-device qubits in cavity qed. *Physical Review A*, 67(4):042311, 2003.

[231] DM Zajac, TM Hazard, Xiao Mi, E Nielsen, and JR Petta. Scalable gate architecture for a one-dimensional array of semiconductor spin qubits. *Physical Review Applied*, 6(5):054013, 2016.

[232] Michael P Zaletel, Roger SK Mong, Christoph Karrasch, Joel E Moore, and Frank Pollmann. Time-evolving a matrix product state with long-ranged interactions. *Physical Review B*, 91(16):165112, 2015.

[233] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. Quantum generative adversarial networks for learning and loading random distributions. *npj Quantum Information*, 5(1):1–9, 2019.