# Communication and Control for Quantum Circuits

*Yatish Patel*

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-77
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-77.html

May 14, 2010

**Communication and Control for Quantum Circuits**

by

Yatish Patel

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

Graduate Division
of the
University of California, Berkeley

Committee in charge:

Professor John Kubiatowicz, Chair
Professor Krste Asanovic
Professor John Flannery

Spring 2010

**Communication and Control for Quantum Circuits**

**Abstract**

Communication and Control for Quantum Circuits

by

Yatish Patel
Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor John Kubiatowicz, Chair

Quantum computers will potentially be able to solve certain classes of problems more efficiently than possible on a classical computer. Due to the fragility of quantum data, a large scale quantum computer will require a robust system to enable reliable communication within the datapath. We present a scalable architecture for a quantum computer which specifically addresses communication concerns. Our design minimizes communication error by using a specialized interconnection network to perform long-distance movement.

We developed a set of tools to construct and study quantum datapath designs based on ion trap quantum technology. Our tools automatically synthesize and insert the interconnection network used for long-distance communication into the target datapath. We present a set of greedy heuristics to optimize the routing and scheduling of communication within this network and show that our approach performs as well as an optimal case determined using integer linear programming. We study a number of different quantum circuits including randomly generated circuits, quantum adder circuits, and ultimately Shor's factorization algorithm and show that designs using our optimizations significantly improve upon prior work in terms of a probabilistic area delay metric.

# Contents

# List of Figures

# List of Tables

# Acknowledgments

None of this work would have been possible without the support of my advisor, John Kubiatowicz, a.k.a. Kubi. Kubi was patient and supportive during my early years when I was hunting for a thesis proposal and helped guide me towards what interested me most. Kubi was never without ideas and whenever I encountered a stumbling block he would invariably have advice to help me tackle the situation.

The quantum CAD flow presented in this work would not have been possible without the contributions from Mark Whitney and Nemanja Isailovic. The three of us put in countless effort into seeing the CAD flow vision realized. Without this joint effort, it would not have been possible to study quantum circuits in such detail. Aside from our professional relationship, I have developed a very close personal connection with Mark and Nemanja. We have gone on numerous adventures together all around the world, ranging from soccer games in Spain to sipping limoncello in Italy to hiking the Great Wall in China. My graduate student career would not have been the same without them.

I'm thankful to my thesis committee members, Professor Krste Asanovic and Professor John Flannery. Both of whom took time out of their busy schedules to provide help and feedback, without which I would never have been able to complete my work.

I started my graduate career sharing a house with Steve Sinha and Sonesh Surana. The three of us all came from Carnegie Mellon and we all decided to enter the CS Ph.D. program at Berkeley the same year. Consequently, our shared experiences have created a bond between us that will last a lifetime. With my family on the east coast, Steve and Sonesh were my family away from home.

Speaking of family, I would not be in the position I am now without the love and support of my parents, brother and sister-in-law and countless "relatives" in the Rochester community. My parents always encouraged me to strive for the best and made many sacrifices to help me get to where I am today.

Additionally, I'd like to thank all of the graduate students in the department that listened to my ideas and provided insightful feedback. Notably, Yury Markovskiy and Victor Wen were always available and willing to listen to practice talks and provided an "outsiders" perspective into my research.

Finally, I would like to thank my girlfriend, Jen. Jen's love and encouragement made it possible for me get through difficult times throughout my graduate career and enabled me to finally complete my work. You are my sunshine.

# Chapter 1

# Introduction

Quantum computers have the potential to perform certain classes of problems more efficiently than a traditional classical computer. The task that garners the most interest in building a quantum computer is the ability to factor large numbers using Shor's factoring algorithm [57]. In order for a quantum computer to tackle such a complex task, it must contain a large number of quantum bits and must be capable of performing many quantum operations. These requirements force us to address a number of concerns, a fundamental one being how best to manage communication of quantum information within the datapath.

Small quantum computer devices have been experimentally demonstrated using a number of different technologies [25, 35, 41, 15]. Common to all these technologies is the inherent fragility of the quantum bits. Simple tasks such as interacting two qubits together, moving a qubit in the system, or even storing a qubit for long periods of time all expose the quantum data to sources of error. To properly execute a quantum circuit we must carefully schedule how qubits move through the datapath to account for these sources of error. Moving quantum bits more than the shortest of distances can introduce enough error into the system to prevent us from obtaining a valid result.

In a quantum computer a qubit is represented by a physical object, e.g., the nuclear and electronic states of a single ion. To perform a two-qubit operation both qubits must be physically adjacent to each other and must move from one location to another. Therefore, during the execution of a quantum circuit qubits must constantly be shuttling around the system to take part in their various operations. Unfortunately, the simple task of moving a qubit can introduce significant amounts of error into the data stored by the qubit if not managed correctly.

One approach to addressing communication issues is to separate movement into two classes: short-distance and long-distance. Short-distance movement encapsulates all direct physical qubit movement and is limited to distances where we can utilize error correction techniques to reduce the error rates. Long-distance moves are those that would be too costly (in error and time) to perform physically, requiring an alternate method of communication. Fortunately, quantum computers allow us to use the process of *teleportation* [6] to perform these types of long-distance moves efficiently.

Teleportation utilizes helper qubits to move quantum information over longer dis-

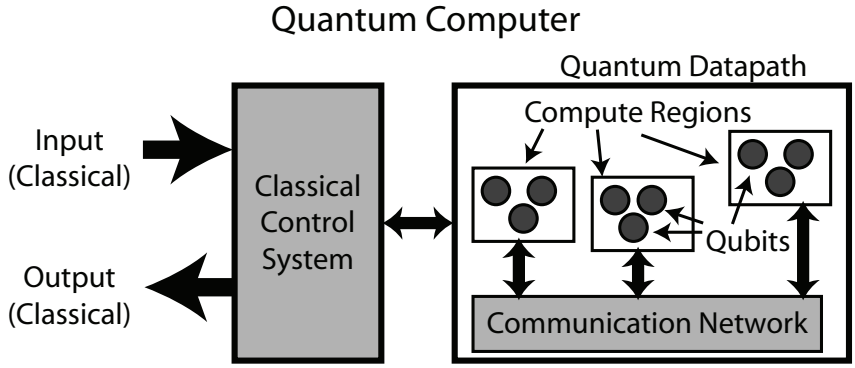



Figure 1.1: High-level view of a quantum computer architecture. The computer ultimately receives and generates classical data, but the main computation is done using qubits in the quantum datapath. The quantum datapath is constructed from a set of compute regions connected via a communication network.

tances. Using this technique we can design an architecture similar to classical multi-core chip with on-chip network: short-distance communication within a "core" (or compute region) occurs ballistically, while long-distance communication (between compute regions) occurs through an interconnection network based on teleportation. Figure 1.1 illustrates this architecture.

The quantum computer is broken up into a classical control portion and a quantum datapath. Both the input data and output data is classical; however, the computation is done internally via quantum operations. The quantum datapath is constructed from a set of compute regions connected via a communication network. The classical control system manages all the operations and orchestrates the full program run.

Variations on this type of architecture have been minimally studied in prior work [42, 66]. The architectures proposed so far however, suffer from a number of limitations:

- The communication networks proposed are high latency. When architectural improvements are made in other parts of the system, network delays start to dramatically impact overall latency.

- The architectural evaluations lacked detail about how various components in the system are constructed. Without these details the architectures cannot be thoroughly evaluated.

- Current research neglects the control system necessary to manage the quantum datapath. The complex support infrastructure and the vast number of qubits contained in the system will require a sophisticated control system that must be addressed.

- General-purpose architectures result in large designs with underutilized regions. In one proposal the authors calculated that a device sized to factor a 1024-bit number would measure approximately $1m^2$ in area [42]. Without optimizing the designs to the target application it will be difficult to build these types of devices.

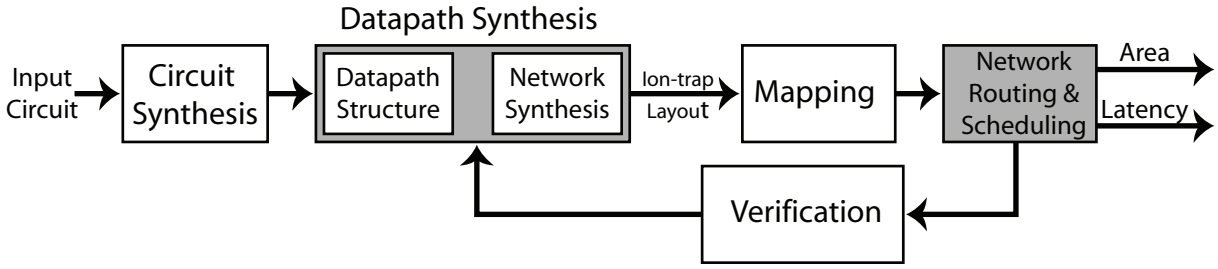



Figure 1.2: Our approach to designing and studying quantum datapaths. The input to the flow is a quantum circuit and the output is a full layout and schedule of operations.

In contrast to a general-purpose architecture, our work is based on using application-specific knowledge to optimize the quantum datapath by concentrating on managing the communication needs. A key result of this work is that by analyzing the quantum circuit we can optimize the communication infrastructure to improve overall circuit latency and reduce sources of error.

Our approach to studying the quantum circuit and performing optimization is shown in Figure 1.2, and consists of the following major components:

**Circuit Synthesis:** Our tools take a quantum circuit specified in a technology independent quantum assembly language combined with technology parameters to synthesize a new circuit with optimized error correction.

**Datapath Synthesis:** We use the synthesized circuit to automatically generate a quantum datapath tailored to the circuits needs. We analyze computation and communication and use this information to determine the best layout of compute regions and network structure.

**Mapping:** The mapping phase maps the circuit operations to the appropriate datapath resources.

**Network Routing & Scheduling:** With a list of operations and locations we generate an optimized routing and scheduling for all the long-distance communication operations that will take place.

**Verification:** The flow outputs a full layout and schedule of operations that are then fed into a verification tool to determine the success probability of the circuit. These results are then used to update the datapath to improve overall area, latency and success probability.

Within this design flow we make the following contributions:

**Short-distance communication:** We develop a method to construct low-level quantum datapaths for arbitrary quantum circuits. We provide tools to schedule gate and movement operations within these datapaths and use them to generate detailed area and communication requirements for our target architecture.

**Long-distance communication:** We present an architecture to manage long-distance communication within large quantum datapaths using a network built on teleportation. We create a control system to manage the complex interactions within the communication network and design low-level datapaths for all the network components. We use these designs to study detailed communication needs for large quantum circuits.

**Routing algorithms for quantum communication networks:** We present algorithms that minimize overhead and delay within the long-distance communication network of quantum datapaths. Our algorithms intelligently determine when and how to preschedule connections given available resources with the primary goal of reducing an overall circuit delay-latency to success probability metric. We show that our adaptive routing algorithm with prescheduling can improve on this metric over the on demand case by a factor of 3 to 4.

In Chapter 2 we present an overview of general quantum computing concepts. In Chapter 3 we outline the Quantum CAD flow we developed that allows us to study various quantum circuits and describe where our contributions fit within this flow. Chapter 4 describes how we build ion trap based datapaths and how communication is managed within such datapaths. Chapter 5 presents our teleportation-based long-distance communication network along with designs for the various network components. In Chapter 6, we describe the routing and scheduling algorithms we use within the long-distance communication network. We conclude in Chapter 7 by using our CAD flow to study Shor's factoring algorithm and its major components.

# Chapter 2

# Quantum Computing

Quantum computers differ from classical computers in how they represent a basic data unit. A quantum computer takes advantage of quantum mechanical properties to implement quantum bits, or *qubits*. Similar to classical data bits, a qubit is capable of representing two values such as 0 and 1. Additionally, a qubit may reside in a *superposition*, or mixture of these two values. The power of quantum computers come from this ability to store data in superposition.

In a classical computer a data register can store a single value at a time, e.g., a 4-bit register can be in one of 16 possible states at any given point. If our goal is to run an algorithm on each of the 16 possible inputs in order to determine a single output we generally have to run the algorithm 16 times. Along these lines, every time we add a new bit to the data register we double the amount of computations we must perform.

Using superposition, a quantum computer can potentially outperform a classical computer at this type of task. Rather than only storing a single value in the data register at a time a quantum data register is able to create a superposition of the values, essentially allowing the register to simultaneously be in all possible states at the same time. While we can compute on all of these states simultaneously, the properties of quantum mechanics dictate that when we measure the register's value we will only obtain a single output value. Because of this limitation, quantum algorithms are structured to increase the probability of reading out the correct answer. Using this technique it is possible to run a quantum algorithm a small number of times in order to determine the desired output rather than once for each possible input value as in the classical case.

## 2.1 Quantum States

As we just mentioned, in classical computing the two possible states a bit can be in is 0 or 1. In a quantum computing the equivalent to these base states are $|0\rangle$ and $|1\rangle$. Unlike classical computing, a quantum state can also be in a linear combination (or "superposition") of these base states where the state is described as $\psi = \alpha|0\rangle + \beta|1\rangle$. An example of a state in superposition is the state $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$.

Internally quantum algorithms take advantage of this state information and can ma-

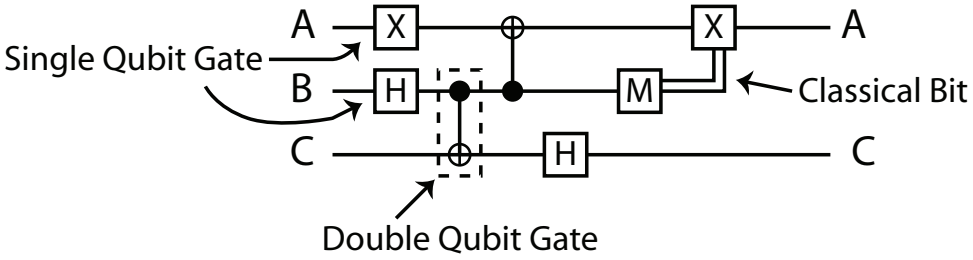



Figure 2.1: Quantum Circuit Model. Small example circuit that operates on three qubits. The sequence of operations is read left to right. Quantum bits are represented by single lines and classical bits are represented by double lines. This circuit contains four single qubit gates, two double qubit gates, and a measure gate. In this example, the measure gate generates a classical bit which is used to control the X gate on qubit A.

nipulate $\alpha$ and $\beta$. However, quantum mechanics prevent us from directly reading out the values of $\alpha$ and $\beta$. Instead, when measuring a quantum bit to determine its state the only possible outcomes are the base state values corresponding to 0 and 1. The value measured is probabilistic and dependent on the values of $\alpha$ and $\beta$. We will obtain the value 0 with probability $|\alpha|^2$ and the value 1 with probability $|\beta|^2$.

Multiple qubits can be composed to create a larger quantum system. For example, the state that describes a two-qubit system is:

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle$$

In this example, the state is a combination of the base states $00, 01, 10, 11$. In this manner a quantum system composed of $n$ qubits is described as a linear combination of its $2^n$ possible base states.

## 2.2 Quantum Circuit Model

One method for expressing quantum algorithms is via the quantum circuit model [46]. In this model, qubits are represented by horizontal lines, while operations are represented by sequences of quantum gates operating on these qubits. Figure 2.1 illustrates a quantum circuit with three qubits, 6 gates, and 1 measurement. Quantum circuits, which are superficially similar to classical circuits, will be our method for expressing quantum logic in this thesis. Two major differences between a quantum circuit and a classical circuit are:

- Quantum gates are unitary and therefore reversible [4]. Reversibility typically requires the use of scratch bits called *ancilla qubits* or simply *ancillae* in order to have the same number of inputs and outputs for each gate.
- Due to the no-cloning theorem [69] qubits cannot be duplicated. This characteristic prevents any fan-out of wires in a quantum circuit.

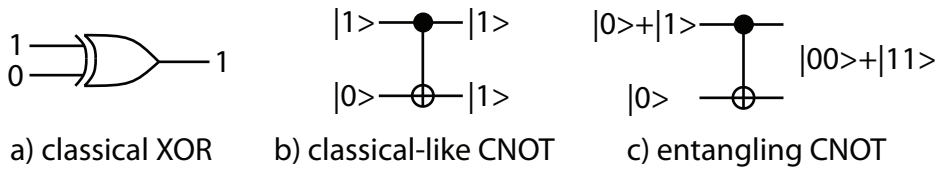



Figure 2.2: A comparison between a classical XOR and its quantum analog: the controlled not or CNOT. The CNOT gate is reversible, thus the additional output. Figure b) outputs the XOR result to the bottom bit. Figure c) shows the same CNOT when the input is a quantum superposition. In this case the output is an *entangled* qubit state, not representable as independent qubit values for the two outputs.

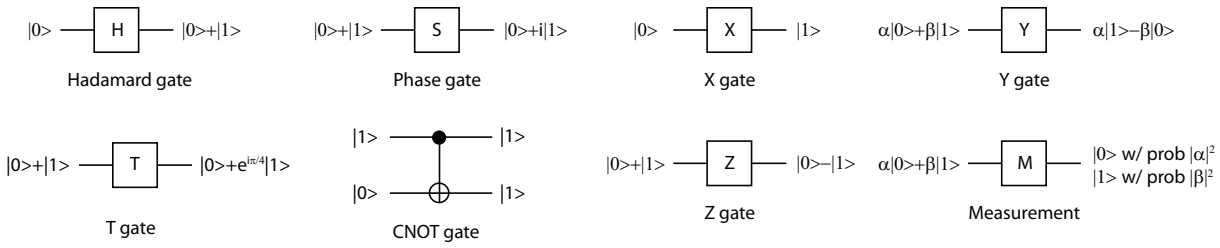



Figure 2.3: Basic gates for quantum circuits which supports a universal quantum computing model. The Hadamard gate converts bit values to phase values and vice versa. The phase, T and Z gates rotate the phase of the "1" qubit value by different angles. The CNOT gate is the same as shown in Figure 2.2 and performs the XOR functionality. The measurement "gate" measures a quantum state, returning a 1 or 0 and collapses any superposition to that value as well. The X is a bit flip, Z a phase flip, and Y a combination of both. The X, Y, Z, and phase gates can be generated by the other gates shown here but we include them since they are often included as physical primitives.

Figure 2.2a shows quantum and classical XOR gates, the quantum version is known as a controlled-not (CNOT) gate as shown in Figure 2.2b. If the quantum inputs are not in a superposition state, the output of the gate is the same as the classical version (with the addition of another output for reversibility). If the "control" input to the CNOT gate is allowed to be a superposition, as in Figure 2.2c, things get more interesting. The resulting output state is an *entangled state* and has no classical analog.

Quantum circuits operate on these entangled superposition states and this is where the power of quantum algorithms comes from. The data cannot stay in superposition indefinitely. In order to read out an answer from the quantum computer, the qubits must be measured so that the data can be presented to the classical world. The process of measurement collapses a superposition state into just one definite bit vector. Measurement also helps us understand the output state from Figure 2.2c. The entangled state $|00> +|11>$ means that when we measure, the resulting classical bit vector will be 00 or 11 (with equal probability).

### 2.2.1 Universal Gates

Due to the more complicated structure of quantum superpositions, there is no single 2-bit universal gate as in the case of the NAND gate in classical logic. Instead, one can use the reversible 3-bit toffoli gate as a universal gate. Since many quantum circuit technologies are practically limited to 1 and 2 bit interactions, we can construct a universal set of 1 and 2 qubit gates as shown in [4]. A standard universal set of 1 and 2 qubit quantum gates was described by Boykin et al. [10] and consists of the CNOT (shown above as a reversible XOR), the Hadamard or H gate (which converts a bit value to a phase value and vice versa), the $\frac{\pi}{4}$ rotation gate, also known as the T gate, and the phase gate. These gates are shown in Figure 2.3 along with some additional gates we will use in the circuits throughout the paper. In reality, different elementary gates are easier or harder to implement depending on which technology one is using.

One more "gate" type is needed: measurement. In order to read out data from a quantum computer, it must be measured, measurement is also instrumental in another useful primitive, known as *teleportation*, which is discussed Section 2.4.1. Measurement is not a unitary gate which is why we do not include it in the universal set, but it is still a necessary quantum operation.

### 2.2.2 Quantum Decoherence

While the power of quantum superposition enables interesting algorithms, it comes at a cost. The sensitivity of a quantum superposition state to noise from the environment cannot be stabilized through noise margins and dynamic feedback as with classical logic. We have to allow a continuum of possible quantum states per qubit instead of 2. For this reason, the error rates of all operations on quantum data are much higher than operations in classical logic. Errors to quantum states cause what is called quantum state decoherence. Error rates in any current demonstrated quantum computing technology are in the range of $10^{-2} - 0.1$ errors per operation. "Realistic" estimates for error rates in the foreseeable future are said to be around $10^{-5} - 10^{-2}$ errors per operation [63] compared to CMOS transistor error rates which range from $10^{-20}$ to $10^{-15}$ errors per gate [56].

The gap is very wide so we would expect to have to pay more attention to errors in quantum circuits than in classical circuits. Indeed, as we mentioned earlier, we have shown that in many cases 95% of quantum circuits will be made up of error correction and fault tolerance modules [34]. With error control circuitry imposing such a large overhead in a quantum circuit, it has not even been conclusively proven that the algorithmic gains promised by quantum computations will not be swallowed up by error correction overhead. The methodical synthesis of circuits with fault tolerance will help us better understand trade-offs between circuit performance and fault tolerance.

### 2.2.3 Fidelity

*Fidelity* measures the difference between two quantum bit vectors. Because of quantum entanglement, each of the $2^n$ combinations of bits in a vector of size $n$ are physically

separate states. For a given problem, one particular vector is considered a reference state that other vectors are compared against. For example, if we start with a bit vector of zeros [0000], and we send the bits through a noisy channel in which bit 3 is flipped with probability $p$, we would end up with a probabilistic vector of $((1-p)[0000] + p[0010])$. The fidelity of the final state in relation to the starting ("error-free") state is just $1-p$. So, in the case of an operational state vs. a reference "correct" state, the fidelity describes the amount of error introduced by the system on the operational state [46]. A fidelity of 1 indicates that the system is definitely in the reference state. A fidelity of 0 indicates that the system has no overlap with the reference state.

In Section 5.2 we use this concept of fidelity to present models for short-distance and long-distance communication.

## 2.3 Error Correcting Codes

Quantum algorithms do not require many qubits to operate, due to their exploitation of qubit superposition. Of course, the use of so few qubits is dependent on a perfect qubit implementation. In reality technologies used to implement qubits are plagued with high error rates. Qubit *fidelity*, or coherence of qubit state, can quickly degenerate to unacceptable levels resulting in data loss. As in classical computing, data redundancy may be used to combat high error rates by way of Quantum Error Correction Codes (QECC) [36, 58, 59]. A QECC encodes a single data qubit used by the algorithm, or *logical qubit*, as some number of basic, unencoded qubits, or *physical qubits*. A single QECC may be applied recursively for more error tolerance at the cost of more redundant qubits. This recursive encoding procedure is called *code concatenation* and is a straightforward way to obtain stronger quantum codes from simpler basic ones.

Once a qubit is represented in the system using multiple physical qubits, performing a gate operation becomes more complicated. A single logical operation is decomposed into a set of physical gate operations, dependent on which QECC is used. Figure 2.4 illustrates the two ways a logical gate may be decomposed into physical gates. In Figure 2.4(a), the logical H gate is performed transversally by applying a physical H to each of the qubits that compose the logical qubit. In Figure 2.4(b), the $\pi/8$ cannot be performed by transversally applying a physical $\pi/8$ gate. Instead, we create an encoded $\pi/8$ ancilla [34] and use X, S, and M gates to perform the logical $\pi/8$ operation. Once the correct set of physical gate operations is performed a logical qubit may undergo error-correction routines to compensate for any physical qubit errors that may have occurred. These error-correction operations involve numerous physical operations and require additional ancilla bits.

There are numerous types of QECCs available for use, each with relative advantages and disadvantages. Designers must account for characteristics such as encoding/decoding complexity, correction complexity, and ancilla requirements when choosing a specific QECC for use within the system.

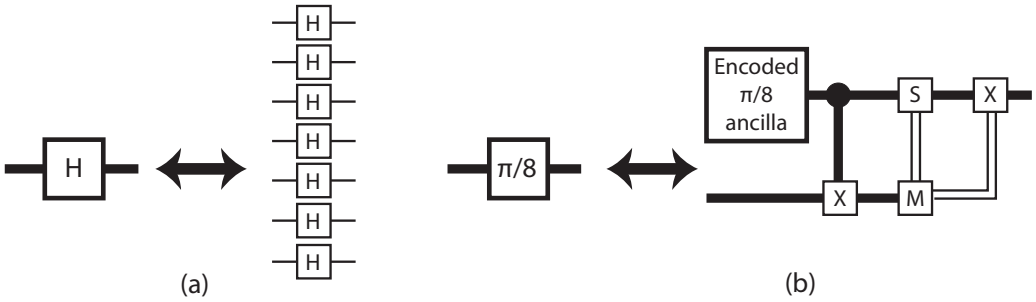



Figure 2.4: Example of a transversal and non-transversal gate. Bold lines represent logical qubits where are encoded by a number of physical qubits. In (a) the H gate is performed on an encoded qubit transversally with a single H gate per physical bit. In (b) the $\pi/8$ cannot be performed transversally. Instead we create an encoded $\pi/8$ ancilla and use X, S, and M gates to perform the encoded $\pi/8$.

## 2.4 Communication

Two qubits must be physically adjacent to perform any two-qubit quantum gate operation. Short-distance movement can be performed using direct physical movement of the qubits from one location in the datapath to another. As with all other quantum operations, qubit movement also suffers from high error rates and result in qubit decoherence and overall move latency proportional to the distance moved. As the size of the quantum datapath grows, the increased distance qubits must travel not only impacts the performance of the computation due to communication time but also introduces significant amounts of error caused by decoherence [47, 33]. Since the data qubits we wish to move are very fragile (even when encoded with a QECC) we cannot rely on direct physical movement for all communication needs. Instead, we separate movement operations into two classes: short-distance and long-distance. Short-distance moves will use direct physical movement, while long-distance moves are performed by using *teleportation*, a technique unique to quantum computing.

### 2.4.1 Teleportation

Within a quantum computer, *teleportation* [6] allows us to move a qubit from one location to another by way of helper qubits. An abstract view of the teleportation process is shown in Figure 2.5. Using this process our goal is to transmit the state of physical qubit D from the source location to a distance target location without physically moving the data qubit (since that would result in too much decoherence). Figure 2.6 shows the circuit representation for this operation (note that E1 and E2 still must be physically separated after the CNOT).

We start by interacting a pair of qubits (E1 and E2) to produce a joint quantum state called an *EPR pair*. Qubits E1 and E2 are generated together and then sent to either endpoint. Next, local operations are performed at the source location, resulting in two

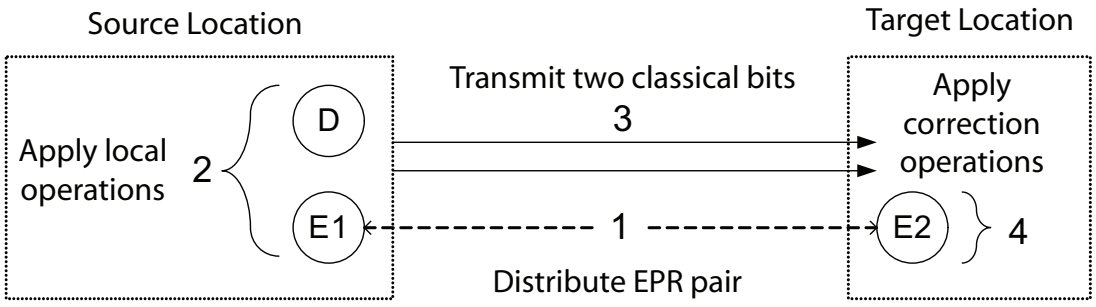



Figure 2.5: Teleporting data qubit D to the target location requires (1) a high-fidelity EPR pair (E1/E2), (2) local operations at the source, (3) transmission of classical bits, and (4) correction operations to recreate D from E2 at the target.

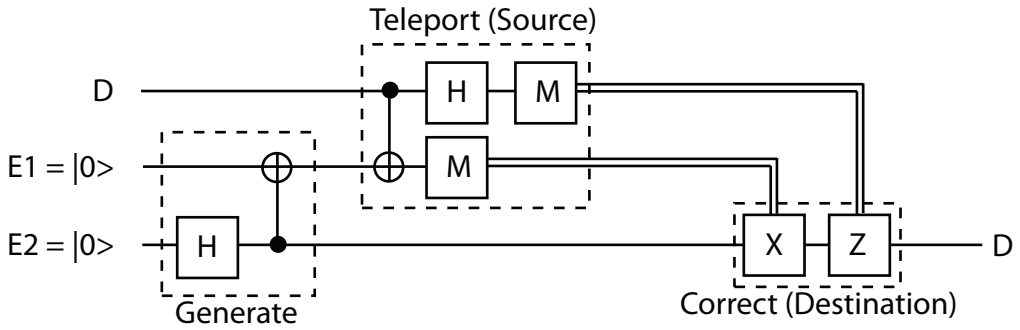



Figure 2.6: Circuit representation for the teleportation operation: The first Hadamard and CNOT gates generate the EPR pair E1 and E2. At the source of the teleportation, one half of the EPR pair is CNOTed with the data followed by a Hadamard and measurements. The measurement results (classical information represented by double bit lines) are transmitted to the destination where they are used to apply X and Z gates to correct E2 and recreate the state of D.

classical bits and the destruction of the state of qubits D and E1. Through quantum entanglement, qubit E2 ends up in one of four transformations of qubit D's original state. Once the two classical bits are transmitted to the destination, local correction operations can transform E2 into an exact replica of qubit D's original state[1]. The only non-local operations in teleportation are the transport of an EPR pair to source and destination and the later transmission of classical bits from source *to* destination (which requires a classical communication network).

We can view the delivery of the EPR pair as the process of *constructing a quantum channel* between source and destination. This EPR pair must be of high fidelity to perform reliable communication. As we will discuss in Section 5.2.3, purification permits a trade-off between channel setup time and fidelity. Since EPR pair distribution can be performed in advance, qubit communication time can approach the latency of classical

[1]Notice that the no-cloning theorem is not violated since the state of qubit D is destroyed in the process of creating E2's state.

communication; of course, channel setup time grows with distance as well as fidelity.

To reliably perform the teleportation operation, the EPR pair distributed to the source and destination must have high fidelity. Since moving the EPR qubits results in the same error build-up as would be encountered when moving the data qubits we cannot use them as is. Instead we utilize a process called *purification*, where two lower fidelity EPR pairs are combined to create a single pair of higher fidelity [22]. In this manner we distribute a number of EPR pairs to the source and destination, purify them down to a single high fidelity EPR pair, and use the resulting pair to teleport the data qubit.

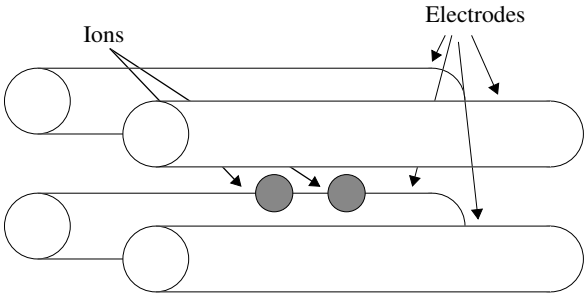
## 2.5 Quantum Computing Technologies

Various technologies exist as promising candidates for building a quantum computer. Physicists have experimentally demonstrated the basic building blocks for quantum computing using technologies such as nuclear magnetic resonance (NMR) [25], solid state nuclear spin [35], superconducting quantum interference devices (SQUIDS) [41], and trapped ions [15, 44]. While each of these technologies are capable of performing single and double qubit operations, most are not convincing candidates for building a large quantum computer due to scalability concerns. Currently, trapped ions show the most promise as a substrate for building a quantum computer big enough to perform useful functions [40, 55]. Consequently, we will focus on designing a quantum datapath using this technology. We start by describing how trapped ion technology is used to implement qubits capable of participating in single and multiple qubit quantum operations. Then we show how these qubits move around the system and outline the procedure for qubit measurement.

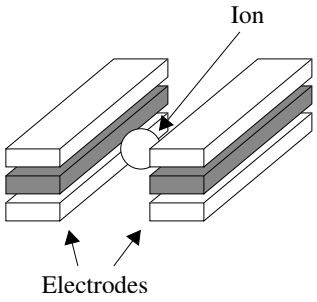
### 2.5.1 Ion Trap and Qubits

In trapped ion technologies, a qubit is represented by the internal nuclear and electronic states of an ion that is trapped within an electromagnetic trap. The trap can be constructed in a variety of different ways, however most implementations are based on the linear RF Paul trap [51, 49] in which a set of electrodes confine ions within the trap. The electrodes utilize a combination of static electric potentials and an RF field to trap the ion. Specific ion species used in the traps vary across implementations; the following ions have all been used in experiments demonstrating quantum computing primitives: $^{9}Be^{+}$ [37], $^{40}Ca^{+}$ [53], $^{111}Cd^{+}$ [30], and $^{88}Sr^{+}$ [11].

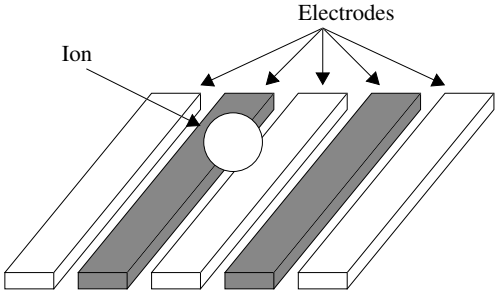
A few diagrams of ion trap implementations are shown in Figure 2.7. Figure 2.7(a) is the initial ion trap construction that used four electrodes to confine ions in the middle area between all the electrodes. Figure 2.7(b) shows the trap built in [30] where a three-layer structure is used to trap ions and Figure 2.7(c) is a planar trap configuration used in [50] in which the ions are trapped above a plane of electrodes. Each of these trap structures present manufacturing and operational challenges if we use them to build large circuits and research is continuously being done to improve on their designs.




(a) Paul Trap




(b) Three-layer Trap

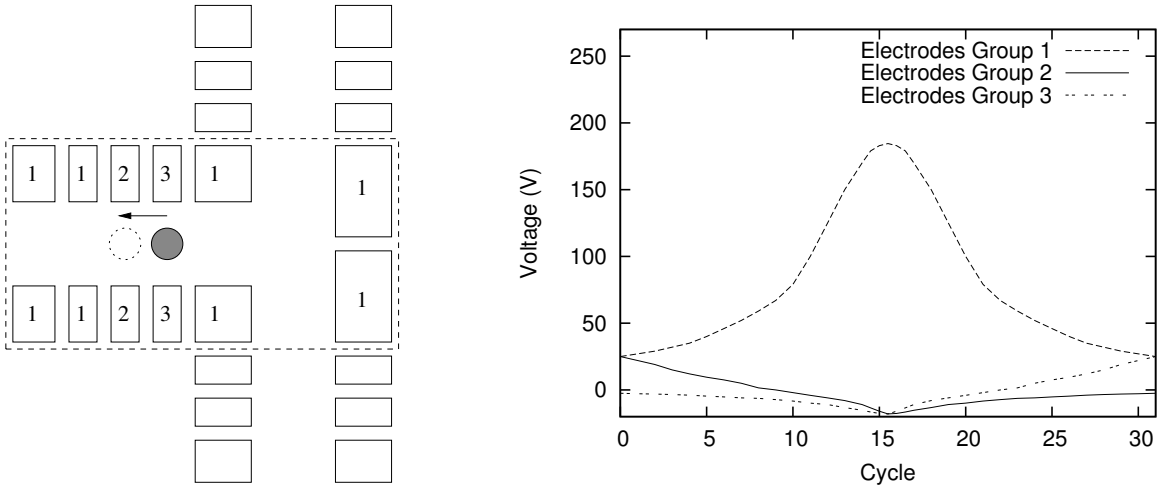



(c) Planar Trap

Figure 2.7: Various structures used to create an Ion Trap. 2.7(a) Original trap configuration where an ion is trapped between 4 cylindrical rods. 2.7(b) Three-layer structure used in [30]. 2.7(c) Planar trap structure from [50], where ions are trapped above a set of electrodes.

| Physical Operation | Error Set 1 [23] | Error Set 2 [63] | Latency in ($\mu$s) [48] |
|---|---|---|---|
| One-Qubit Gate | $10^{-6}$ | $10^{-4}$ | 10 |
| Two-Qubit Gate | $10^{-6}$ | $10^{-4}$ | 100 |
| Measurement | $10^{-6}$ | $10^{-4}$ | 500 |
| Zero Prepare | $10^{-6}$ | $10^{-4}$ | 510 |
| Straight Move ($\sim$30 $\mu$m) | $10^{-8}$ | $10^{-6}$ | 10 |
| 90 Degree Turn | $10^{-8}$ | $10^{-6}$ | 100 |
| Idle (per $\mu$s) | $10^{-10}$ | $10^{-8}$ | N/A |

Table 2.1: Error probabilities and latency values used by our CAD flow for basic physical operations




(a) Electrode groups.

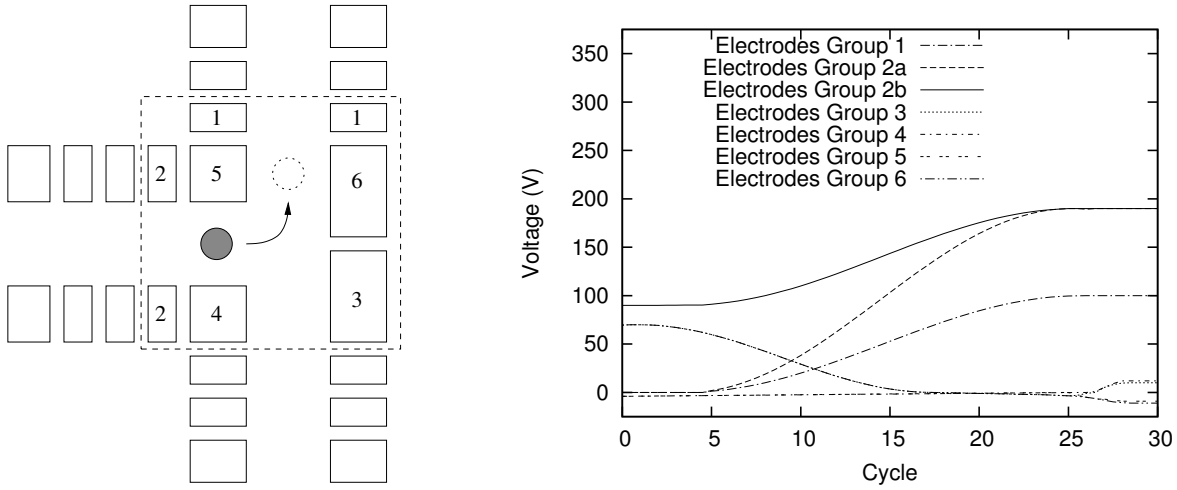
(b) Electrode voltages.

Figure 2.8: Electrode groups and voltages used to move a qubit from one trap to another. Each electrode group shares the same voltage level. Electrodes outside the dashed box are not utilized in the movement protocol.

### 2.5.2 Movement

As described in the previous section two qubits must be physically adjacent to each other in order to participate in a double qubit gate operation such as a CNOT. The ion trap substrate must be capable of moving qubits around the system to enable us to perform these gate operations. At rest, the qubits are held within a trap using electric potentials provided by the surrounding electrodes. If we change the potentials on the electrodes, we can effectively encourage the qubit to move out of its current trap and into a neighboring trap in both 1 and 2-dimensions [30, 50].

As an example, movement of a qubit can be thought of as involving the following operations: (1) lowering the voltages on the destination trap to attract the qubit, (2) raising the voltages behind the qubit to push it out of its current trap and then (3) establishing the correct voltages to confine the qubit to the new trap. In reality, the voltages applied to the electrodes are significantly more complicated than this example and vary depending on the ion species used and trap configuration. To minimize error introduced into the ion's internal state via motional heating, adiabatic movement protocols requiring precise control of the potentials on the trap electrodes are used. The following sections summarize the movement protocols experimentally demonstrated in [30, 32, 70], in which the authors used an array of 11 ion traps to create a T-junction shape. The individual traps were built using a three-layer structure as shown in Figure 2.7(b). The figures in this section are simplifications of the experimental setup and only show the electrodes as viewed from above.




(a) Electrode groups

(b) Electrode voltages

Figure 2.9: Electrode groups used to move a qubit around a corner. Electrodes outside the dashed box are not utilized in the movement protocol. All electrodes in a group use the same voltage level. For group 2, 2a is the signal for the top layer, and 2b is the signal for the bottom layer.

### Straight Line Movement

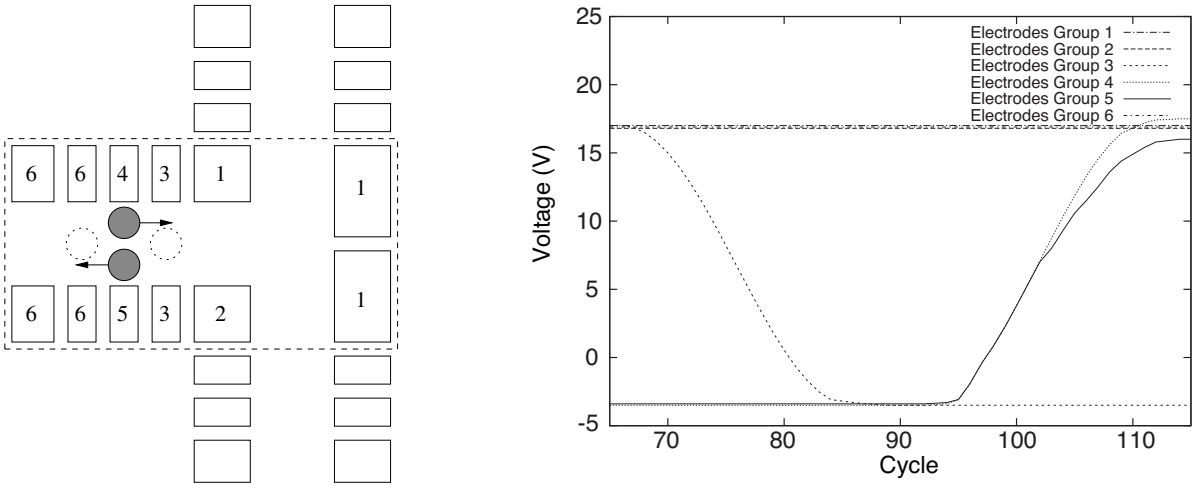
Figure 2.8(a) and Figure 2.8(b) show the electrodes and voltages used to move a qubit to an adjacent trap. For the first half of the protocol the electrodes behind and far in front of the qubit are raised while the electrodes next to and directly in front of the qubit are lowered. Then the outer electrodes are lowered while the old trap electrodes are raised allowing the qubit to move into the new trap.

### Corner Turning

Figure 2.9(a) and Figure 2.9(b) show the electrodes and voltages used to move a qubit around a corner. A far more complicated sequence of voltages is required in order to move an ion in two dimensions as compared to moving a single trap in one dimension. The ion is first pushed into the junction region and then guided into the new trap around the corner.

### Splitting and Combining

Figure 2.10(a) and Figure 2.10(b) show the electrodes and voltages used to split apart two qubits that occupy the same trap. At the end of the protocol, the qubits reside in two separate traps. Combining two qubits into the same trap is performed using a time reversal of the splitting protocol.




(a) Electrode groups

(b) Electrode voltages

Figure 2.10: Electrode groups and voltages used to implement ion splitting and combining. Two ions start in the trap region between electrode groups 4 and 5 and move apart. Electrodes outside the dashed box are not utilized in the movement protocol. Time reversing the protocol will combine two separated ions into the same trap.

### 2.5.3 Measurement

To determine of the state of a qubit we use a measurement procedure specific to the ion species used. The procedure involves one or more lasers and a measurement device. The lasers are applied to the ion in order to excite a transition in the internal state. Depending on the starting state of the ion the change in internal state may generate fluorescence which is detectable by the measurement device. Since only certain state changes generate fluorescence, one starting state can be distinguished from another by the presence or lack of this fluorescence.

As an example, the procedure described in [54] uses two lasers and a CCD camera to detect the state of a qubit represented by $^{40}Ca^{+}$. One laser is at 397nm and the other at 866nm. The CCD camera detects any fluorescence that occurs during the application of the lasers. The measurement procedure takes 10ms and was able to discriminate the state of the ion with close to 100% efficiency.

### 2.5.4 Gate Operations

As in measurement, performing gate operations in ion trap technology is conducted using laser pulses. By controlling the frequency and phase of the laser and the duration of the laser pulse, the ion's internal state can be manipulated to perform single qubit gate operations [37, 52, 5, 53].

From this listing, we note that each gate could take about 3 or 10 separate consecutive laser pulses, depending on whether it is a single or double qubit gate. Each of these pulses must be applied for a reasonably precise amount of time. A description of the qubit ion

energy state transition curves under laser application is shown in [28] and [53]. The important thing to note here is that qubit values are oscillatory in the time evolution under laser application, thus the amount of time the laser pulse is applied is critical in performing the correct gate. The approximate oscillation frequency of the ions used in many of these experiments is around $200\mu s$, thus in order to maintain a gate error of less than $10^{-4}$ or so, we would need to control laser pulse length to a resolution of roughly $200\mu s \times 10^{-4} = 20ns$.

In addition to precise laser pulse length, substantial optics are necessary to sufficiently focus the laser to a narrow enough beam width in order to address individual ions within the trap. As mentioned in [45], two qubit gates require qubit ions to be adjacent in a single ion trap with a distance between them around 7-20$\mu m$, leading to a requirement of a beam width of around $5\mu m$. In this particular experiment, obtaining such a resolution was achieved with a rather large Nikon lens. Also mentioned in [45] is the need for a laser with a very stable frequency, one that is within 1-kHz of the precise transition frequency between qubit ion energy levels. This limitation precludes the use of miniaturized semiconductor laser diodes from any current fabrication technology.[2]

Due to the large size (and probably expense) of the gate lasers and focusing optics, we see a strict resource limitation on the number of laser beams we can produce for our quantum computer. Additionally, double qubit gates require up to 4 different frequencies of laser light, so in order to perform a single gate, we may need 4 large laser apparatuses. The one thing we *can* miniaturize is an optical system to divert and split the already focused and stabilized laser beam to deliver them to the particular trap locations. The technology of electro-mechanical micro mirrors has already been applied on a large scale to commercial optical routing technologies [8] and is capable of deflecting beams with over 1000 individually addressed micro mirrors.

For the above reasons, we assume a small number of lasers and a very large and flexible system for routing and aiming the limited number of lasers. This structure naturally lends itself to a SIMD design with individual laser beams being split and routed to many trap locations, allowing a single gate type to be applied at many locations simultaneously using one laser. A SIMD design imposes a globally synchronous model of operation at the lowest level where large numbers of physical gates require synchronization to be performed by a limited number of lasers.

### 2.5.5 Abstraction

The various ion trap implementations described in Section 2.5.1 differ slightly in various details such as ion species and electrode configuration (planar versus non-planar). Many of these low level details do not effect our study of the datapath and control system and therefore we use a simple model to represent all of these ion trap implementations: a

[2]Semiconductor lasers have beams consisting of multiple frequencies due to a large number of available modes just above and below the band gap where the electrons and holes recombine. Additionally, the band structure is highly sensitive to local temperature and current fluctuations, meaning that even if a single mode could be isolated, that particular mode would fluctuate by an unacceptable amount [29].

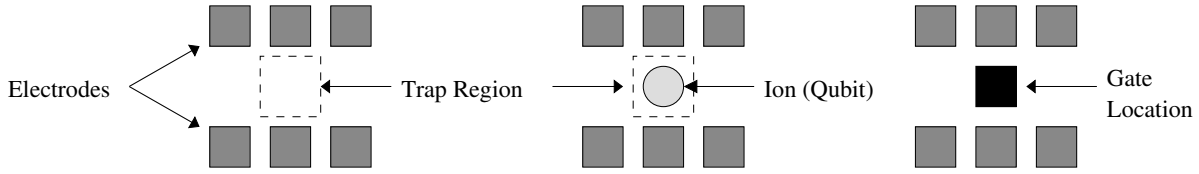



Figure 2.11: Ion trap abstraction. The dark gray boxes represent electrodes, and the dashed box is a trap region capable of holding an ion (qubit). The left side of the figure is an unoccupied ion trap, and the right side shows a trap with an ion confined in it.

set of electrodes used to confine an ion. For the rest of our study we will think of ion traps in this fashion, representing an ion trap as in Figure 2.11, where electrodes surround a region capable of trapping one or more ions (a larger library of our low-level modules is shown in Figure 3.6).

In our abstraction, electrodes are depicted by the gray boxes and the trap regions between electrodes that hold ions are shown as dashed boxes. As mentioned earlier, movement only requires manipulation of the electrode voltages, however, gate operations require additional laser resources. These laser resources may not be accessible at every trap region and therefore we specially mark trap regions capable of performing gate operations with a black box.

Our simple abstraction allows us to conduct our study without being constrained to a specific ion trap implementation with explicitly defined electrode layouts, sizes and spacings. The model insulates us from some of the extremely low-level details, while still revealing the control complexities that must be addressed in the design of an ion trap-based quantum computer. However, it is important to note that our layout abstraction is chosen to be sufficiently accurate (based on existing designs) that it can be used for area and latency estimation.

While our model allows us to study a wide variety of datapaths it has a few limitations. Recently, some ion-trap structures have been proposed that use 60° intersections [9, 2]. Our model currently only supports 90° intersections and would need to be modified to study datapaths built using these structures.

### 2.5.6 Comparing to Classical CMOS

Some key differences between this quantum circuit technology and classical CMOS are as follows:

- "Wires" in ion traps consist of rectangular channels, lined with electrodes, with atomic ions suspended above the channel regions and moved ballistically [40]. Ballistic movement of qubits requires synchronized application of voltages on channel electrodes. Thus, each wire requires movement control circuitry to handle any qubit communication.

- A by-product of the synchronous nature of the qubit wire channels is that these circuits can be used in a synchronous manner with no additional overhead, enabling

some convenient pipelining options. For example, moving a group of qubits down a channel can be accomplished by sharing the control signals for each single qubit move in the path, rather than requiring independent control signals for each qubit in the group.

- Each gate location will likely have the ability to perform any operation available in ion trap technology which enables the reuse of gate locations within a quantum circuit.

- Scalable ion trap systems will almost certainly be two-dimensional due to the difficulty of fabricating and controlling ion traps in a third dimension [31]. This limitation means that all ion crossings must be intersections.

- Any routing channel may be time shared by multiple ions as long as control circuits prevent multi-ion occupancy and only allow a single ion in the channel at a given time. Consequently, our circuit model resembles a general network, qubits can be likened to data packets that are routed through a network of ion traps.

- Movement latency of ions is not only dependent on Manhattan distance but also on the geometry of the wire channel. Experimentally, it has been shown that a right angle turn takes substantially longer than a straight channel over the same distance [50, 30].

# Chapter 3

# Quantum CAD Flow

The overall goal of this work is to facilitate the design and evaluation of large quantum circuits with emphasis on addressing their communication and control needs. Most study has been limited to extremely small quantum datapaths on the order of 10s of qubits. While it is interesting and worthwhile to study these smaller datapaths, we must also keep our sight on our most sought after goal: a quantum computer capable of performing tasks such as Shor's factoring algorithm on large input sizes. Previous research on large-scale quantum architectures has mainly relied on manually specifying datapaths with the goal of creating a general-purpose quantum computing device. Unfortunately this approach doesn't lend itself well to more thorough study of quantum datapaths. For example, studying trade-offs in low-level datapath design and communication network design is greatly simplified when automated tools are available. In order to conduct a comprehensive study of quantum circuits we created a computer-aided design (CAD) flow for quantum circuits.

We will show how the process of taking a quantum circuit description and creating a physical layout is similar to the classical case. We will start by discussing some of the differences between classical and quantum circuits:

**Fault frequency:** Errors are many orders of magnitude more likely in quantum logic than in classical which places an additional requirement on a quantum circuit for very strong fault tolerance.

**Classical control:** Implicit in everything mentioned so far is a network for controlling qubit motion and gate operation. We address some of the problems of *classical control synthesis* in this CAD flow. Along with any physical layout of a circuit, we also generate a control schedule to implement the movement and gate operations on the qubits.

**Gate reuse:** In a classical layout, performing a NAND followed by another NAND requires two physical NAND gates connected by some wires to transmit the data value. In contrast, a quantum circuit can time multiplex a single physical gate location to perform the two gates in sequence without moving the data. This feature reduces movement (reducing movement error) and reduces area, making fabrication easier.

**Reversibility:** The reversibility constraint on quantum logic requires the use of many more ancillary qubits to be created and tracked throughout the course of the computation.

In addition to the differences in the quantum and classical circuit models, there are differences in the underlying technology used to lay out our circuits. As mentioned in Section 2.5, we focus on ion trap quantum computing in this study and so to compare ion traps with classical CMOS:

**Bit persistence:** In ion traps, qubits are physical entities that cannot simply be created or dissipated after the value on them is no longer useful. Dead physical qubits must be disposed of or recycled, and new qubit values must be allocated a new physical ion. The requirement for having many ancilla qubits to implement reversibility has a large impact on this requirement because many qubits are created and destroyed in a quantum circuit.

**Planar wiring:** Qubit ions are suspended in a vacuum above the electrodes and must have space to float along surface channels, therefore it is unlikely there will be more than one layer of ion trap "wiring" on the fabricated chip. Thus, all wiring crossings are actual 4-way intersections where only one direction can be operational at a time. This limitation impacts the area used to place dedicated channels for ion movement, as well as scheduling of ions along potentially shared channels.

**Multiplexing resources:** Since qubits have a physical extent, different qubit values can share a channel/wire in a circuit as long as they are spaced far enough apart to limit unanticipated interactions. Thus, wires can be multiplexed rather easily, which is important since the number of wires is limited to what we can fit in the plane.

**Communication-cost metrics:** Strict Manhattan distance is not an accurate measure of wire length because preliminary studies have shown that turning corners and traversing intersections will be more time consuming and acquire more vibrational heating (and errors) than moving straight through a one-way channel [50, 30].

The quantum CAD system we have developed is modeled after a classical CAD tool flow, but accounts for many of these differences between quantum and classical circuits. Figure 3.1 shows the currently available components in our CAD flow. As mentioned earlier, our tools rely on a relatively simple input specification and do not currently do much circuit synthesis from higher-level descriptions. Our toolset is "bottom heavy" in that we are more interested in creating a detailed physical design that implements simple gate-level circuits than we are in transforming and optimizing high-level quantum algorithms.

The components highlighted in grey are the focus of this work and will be covered in great detail, but for now, we will give a brief description of all the components.

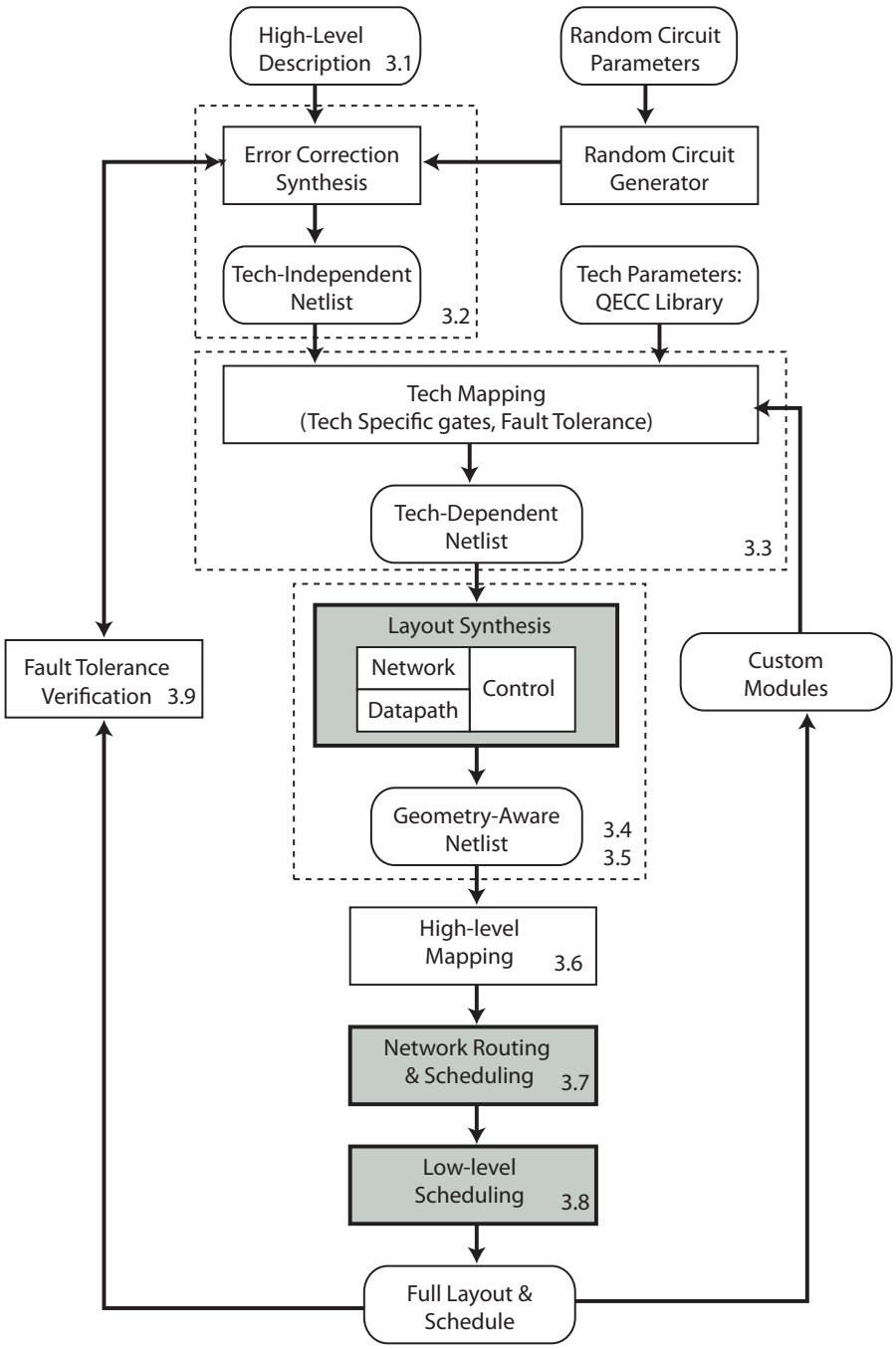



Figure 3.1: A high level view of our computer-aided design flow for quantum circuits. The highlighted blocks denote the contributions focused on in this work. Dashed boxes correspond to the indicated section number where details of the stages are explained.

| Category | Name | Errors? | # of (cla/qu)bits | Description |
|---|---|---|---|---|
| Pure Quantum | h | yes | 1 | Hadamard gate, translates between X and Z basis. |
| | x | yes | 1 | Bit flip. |
| | y | yes | 1 | Bit and phase flip. |
| | z | yes | 1 | Phase flip. |
| | s | yes | 1 | Phase gate: phase rotation by $\pi/2$. |
| | t | yes | 1 | T gate: Phase rotation by $\pi/4$. |
| | cx | yes | 2 | CNOT gate: controlled-X gate, bit flip on target based on control. |
| | cz | yes | 2 | Controlled-Z gate, phase flip on target based on control. |
| | cphase | yes | 2 | Controlled-phase gate, phase rotation by $\pi/2$ based on control. |
| | xprepare | yes | 1 | Prepare input qubit in a particular state in the X basis. |
| | zprepare | yes | 1 | Prepare input qubit in a particular state in the Z basis. |
| | correct | no | 1 | Logical-only operation representing a correction step, encoded gate implementation is code dependent. |
| Pure Classical | or | no | variable | Set output bits based on logical or over all input bits. |
| | xverify | no | variable | Verify that there are no X errors on the classical syndrome bits, sets output bit if there are errors that are not undetectable by the code. Exact syndrome check is code dependent. |
| | zverify | no | variable | Verify that there are no Z errors on the classical syndrome bits, sets output bit if there are errors that are not undetectable by the code. Exact syndrome check is code dependent. |
| Quantum-Classical | xmeasure | yes | 2 | Sets classical bit based on quantum bit value in the X basis. |
| | zmeasure | yes | 2 | Sets classical bit based on quantum bit value in the Z basis. |
| | xcorrect | no | variable | Corrects bit flip (X) errors on input qubits based on the values of input classical bits. |
| | zcorrect | no | variable | Corrects phase flip (Z) errors on input qubits based on the values of input classical bits. |
| | (predicate) | no | variable | Execute given quantum or classical operation if list of predicates are all satisfied. |

Table 3.1: Summary of all the quantum instructions we use. Pure quantum instructions input quantum data and output quantum data. Pure classical instructions manipulate classical data only. Quantum-classical instructions either use classical data to manipulate the quantum data, or measure quantum data to generate classical output.

## 3.1 Application Circuit Specification

The primary method for input of application circuits into the CAD flow is the use of the QASM description language. The original QASM was first introduced by Balensiefer et al. in [3]. Basic quantum operations and qubit operands, similar to classical registers, are listed in the order in which they are supposed to be executed.

The full QASM instruction set that we use is shown in Table 3.1. The instructions that do not introduce errors are *virtual* instructions used for bookkeeping of qubit states or classical information and do not correspond to actual physical quantum operations. We note that the `x/zcorrect` operations are not error prone because they are only virtual instructions that do qubit error state updates. They do not correspond to the entire error-correction process which contains many error-prone physical gates.

Figure 3.2 shows an example of a quantum circuit and its QASM specification. In this

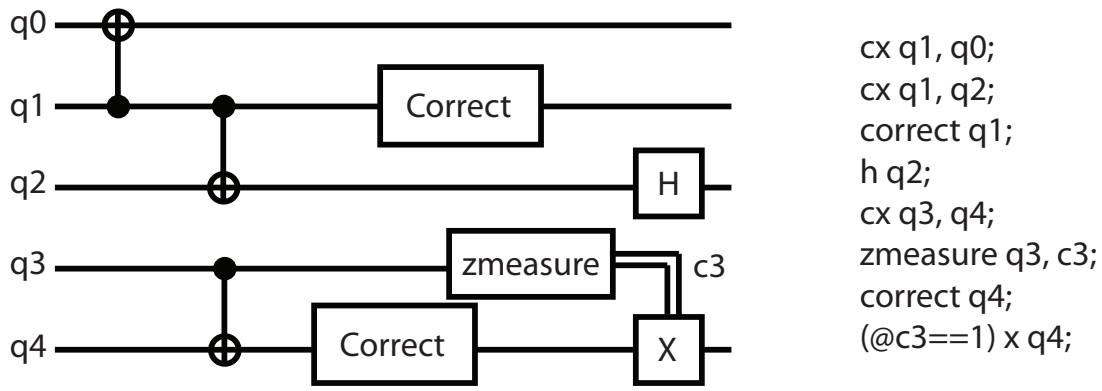



Figure 3.2: A quantum circuit and the equivalent QASM instruction stream representing it.

example gates/instructions are read from top to bottom in order, thus gates dependent on the output of earlier gates appear later in the instruction stream. Two qubit gates like the CNOT or `cx` (controlled-X) take the names of 2 qubit registers, the first one being the control and the second, the target. Correction gates operate on a single logical qubit. A measurement gate takes in a qubit and outputs a classical bit; classical bit register names typically starting with a `c`. In this example `c3` is a classical bit measurement outcome which then predicates the execution of the last `x` gate. Predicates only compare classical bits to a constant value (no quantum bits) and determine whether the gate they are guarding is executed. So in this example, if the zmeasure outcome sets `c3` to 1, then the `x` gate will be applied to `q4` later.

In a QASM definition of a circuit we explicitly declare qubit state and quantum gates. Here is an example of a 1-bit quantum adder in QASM:

```
1  qubit cin;
2  qubit cout;
3  qubit a;
4  qubit b;
5
6  input cin;
7  input a;
8  input b;
9  toffoli cout, a, b;
10 cx b, a;
11 toffoli cout, cin, b;
12 cx b, cin;
13 output a;
14 output b;
15 output cout;
```

The program starts with a declaration of qubit states or "registers" declared with the *qubit* keyword. These states will later be mapped to a physical element that can represent a 2-level quantum system. The qubit declarations are followed by a sequence

of gate operations on the qubit states. In this example, we use 3 qubit `toffoli` gates and 2 qubit `cx` (CNOT) gates. This circuit takes a carry-in bit, `cin` and two input bits, `a` and `b`. The sum output is in `b` and the carry-out is in `cout`. We also use the special purpose virtual instructions `input` and `output` to specify which qubits would be set as input/output for the circuit

In addition to simple sequences of gates, we can specify hierarchically structured programs through use of our added support of *modules*. We can define a sequence of qubits and gates as a module and instantiate it in multiple places throughout the program. For example, here is a circuit for a 4-bit adder made out of 1-bit adders:

```
1  module carry cin, a, b, cout {
2    toffoli cout, a, b;
3    cx b, a;
4    toffoli cout, cin, b;
5  };
6
7  module carry_inv cin, a, b, cout {
8    toffoli cout, cin, b;
9    cx b, a;
10   toffoli cout, a, b;
11 };
12
13 module sum cin, a, b {
14   cx b, a;
15   cx b, cin;
16 };
17
18 qubit cin0, cin1, cin2, cin3, cout;
19 qubit a0, a1, a2, a3;
20 qubit b0, b1, b2, b3, b4;
21
22 carry cin0, a0, b0, cin1;
23 carry cin1, a1, b1, cin2;
24 carry cin2, a2, b2, cin3;
25 carry cin3, a3, b3, b4;
26 cx b3, a3;
27 sum cin3, a3, b3;
28 carry cin2, a2, b2, cin3;
29 sum cin2, a2, b2;
30 carry cin1, a1, b1, cin2;
31 sum cin1, a1, b1;
32 carry cin0, a0, b0, cin1;
33 sum cin0, a0, b0;
```

Note that when calling modules, the register names must effectively be renamed so the physical element with the qubit state of `b3` must be renamed `b` in the `sum` module. We will discuss this issue later when we discuss tracking qubit error state.

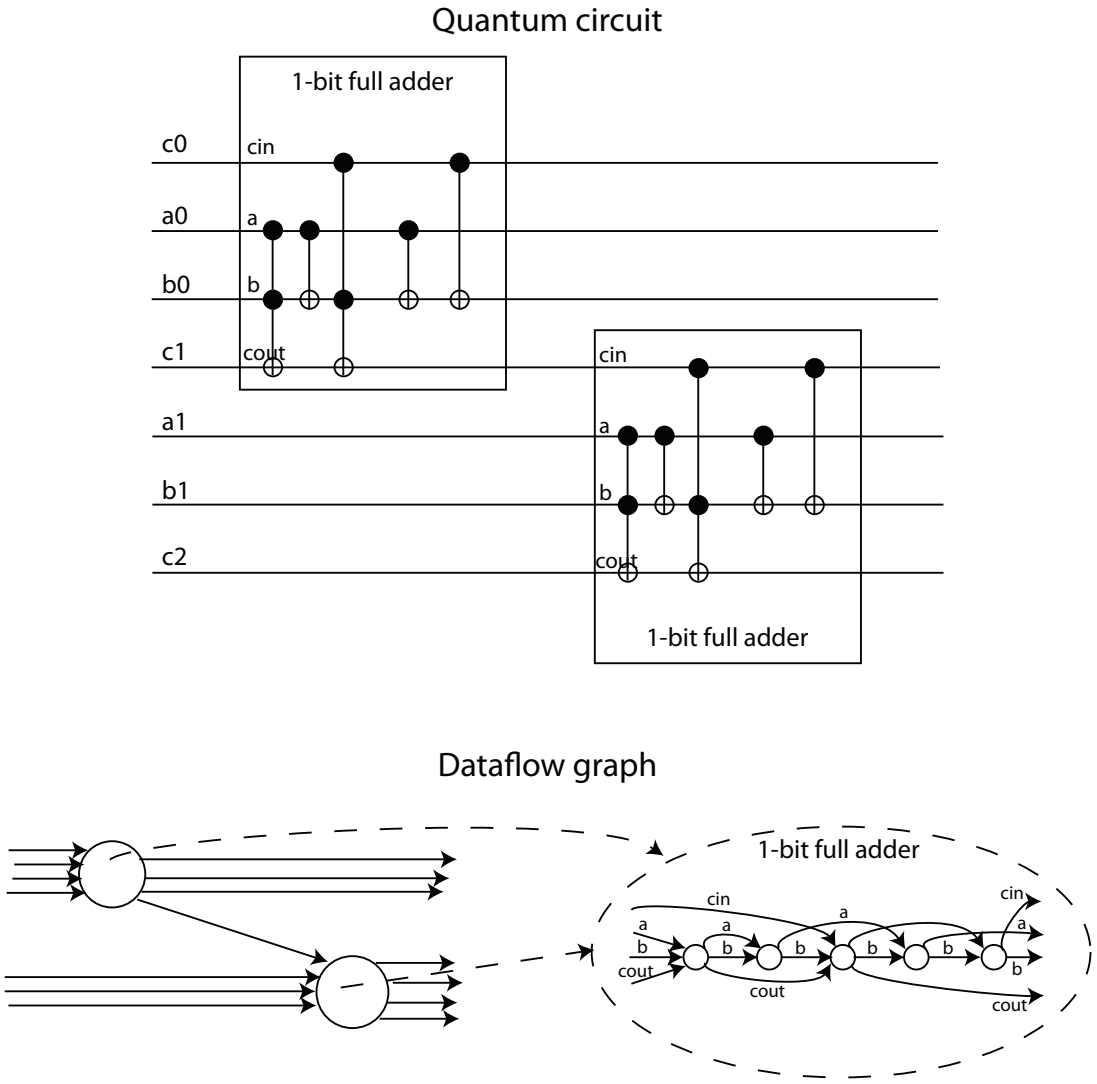



Figure 3.3: Gate networks are represented as linked, modular dataflow graphs. In this example, the top level graph consists of two nodes that each correspond to a 1-bit full adder. They both refer to the 1-bit full adder module dataflow graph.

### 3.1.1 Application Dataflow Graph

Our core data structure representing the input application logic is a hierarchical, annotated dataflow graph. Figure 3.3 shows an example of such a graph. In this example, the top level graph that consists of a 2-bit ripple carry adder is implemented with 2 nodes that both point to the same full adder graph.

We maintain the hierarchy of the dataflow graph throughout the various stages of the CAD flow. For instance, when we are encoding a quantum circuit into a quantum error correction code (QECC), each logical gate is represented by a module pointing to a graph that represents a specific encoded version of that gate type. If we are concatenating

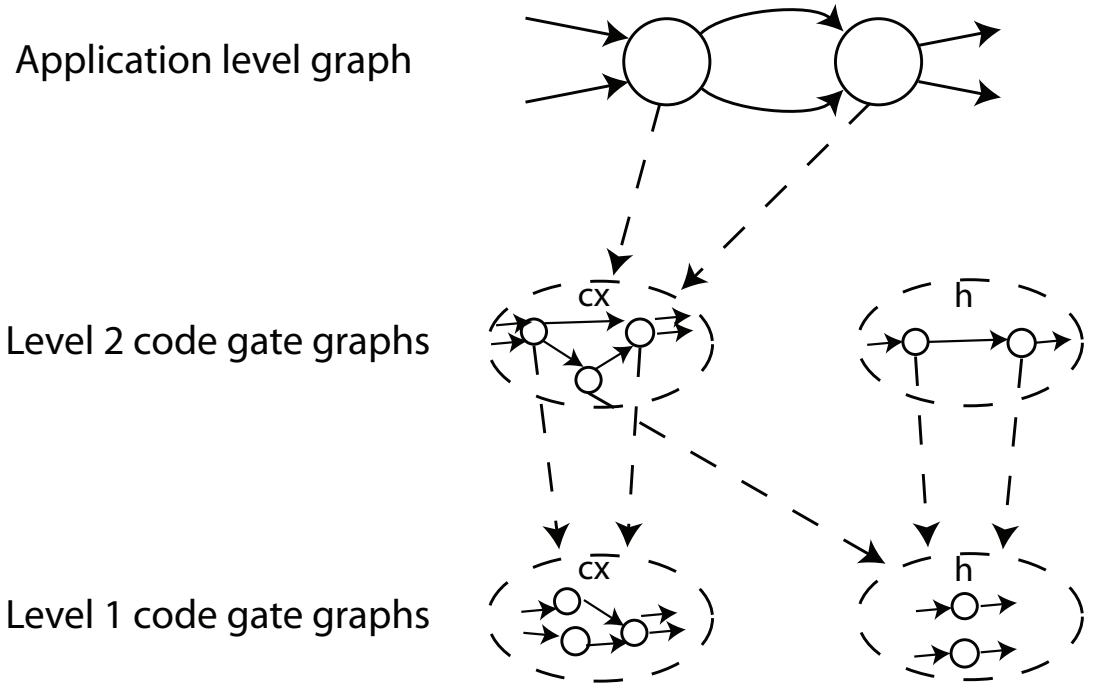



Figure 3.4: Hierarchical dataflow graphs are used to represented different levels of QEC encodings. In this example we have the 2 gate application circuit encoded in 2 levels of codes. Each code has a library of graphs, each graph implementing an encoded version of one gate type.

several codes together to yield more reliability, there might be multiple levels in the hierarchy for gate implementations in different codes, as shown in Figure 3.4. The modular representation of a QECed circuit is especially beneficial since fault tolerant subcircuit substitution introduces orders of magnitude more gates (about 500x for a one level [[7,1,3]] code, for example). Since most of the subcircuits are the same, mapping all logical gates of a particular type to a single graph for the fault tolerant version makes our design representation tractable for large circuits.

Additionally, we may have different elementary gates that can be performed physically depending on the implementing technology. We enable the technology-specific translation by providing technology gate libraries to translate logical gates into physically implementable gates. Our technology translation currently converts single logical gates to groups of technology-dependent gates so we utilize the hierarchical nature of our dataflow graph again to maintain a modular mapping mechanism.

Note that even though only a single instance of a module is created and stored for a particular graph. When we traverse the graph, we must re-traverse the single module dataflow graph for all the nodes of a particular module type. This process adds some complexity to the traversal of our modular graphs.

## 3.2 Error Correction Circuit Optimization

Our tools allows us to vary the amount of error correction that occurs in the circuit. A conservative approach is to insert an error correction operation after every logical gate in the circuit. This approach is fast but it doesn't perform any circuit based optimizations and can result in circuits unnecessary correction operations, increasing area needs and total circuit latency.

We optimize the error correction process by using a Retiming approach detailed in [68]. In any non-trivial circuit some qubits will undergo more gates, movement, of idling than others. Thus, different qubits will have different probabilities of error at different times throughout their life in the circuit. The previous conservative approaches to error correction call for the assumption that each logical qubit be corrected after every logical gate. Thus, it is effectively treating all qubits in the circuit as if they have the same probability of having an error at all times. This assumption is not the case and therefore the treatment is overly conservative.

Our approach effectively analyzes each qubit at each gate and applies error corrections only when necessary. We draw an analog between minimizing latency in synchronous classical circuits and minimizing failures in our quantum circuits. We use the technique of circuit retiming [38] to "recorrect" the given circuit. Based on an approximation of how errors propagate in a circuit, we can more effectively distribute error correction steps throughout our circuit.

## 3.3 Quantum Logic Synthesis

The primary goal of logic synthesis in classical CAD flows is to derive a technology-dependent gate network from a high-level circuit specification. In addition to this goal, our quantum logic synthesizer also must add additional circuitry to ensure that our circuit is fault tolerant.

### 3.3.1 Technology Dependent Gates

Since we allow the superset of all interesting quantum gates from quantum computing literature to be used in our QASM definitions, we have a synthesis stage in which we convert QASM gate operations into gate operations that are supported natively by the type of quantum computing technology we are designing for. We specify *technology libraries* to map abstract QASM gates to physically implementable gates for each technology our CAD flow can target. For example, since we limit the number of qubits in an ion trap to 1 or 2 per interaction, we cannot physically implement a toffoli operation, so instead we translate toffolis into a sequence of 1 and 2 qubit gates from the ion trap technology library.

### 3.3.2 Fault Tolerant Gate Constructions

Once we have a set of physically implementable gates to work with, we must next make them fault tolerant. We can apply quantum error correcting codes to the problem, transforming each logical gate from the technology-dependent network into an encoded subcircuit implementing the same operation fault tolerantly. For each code our CAD flow supports, we have a library of encoded gates that can be substituted into the circuit. These libraries are generated automatically using Andrew Cross's ftqctools [16]. The selection of QECC to be used in the synthesized circuit is selected by the user.

## 3.4 Datapath Microarchitectures

The first step in specifying overall spatial placement of computation elements for a quantum circuit is to have a high-level organization of the different quantum computer elements. Our solution to this problem builds on several previous works that focused on *tiled-dataflow architectures* for a quantum computer.

Proposed architectures, including ours, have consisted of computation regions connected by an interconnection network using quantum teleportation [42, 33]. High fault rates in quantum computing necessitate the widespread use of quantum error correction (QEC). Further, ancilla state generation is important to aid in the correction process [61] and as an integral part of quantum algorithms, as we showed in [34].

### 3.4.1 Three Major Organizations

Figure 3.5 shows three major *datapath organizations* that represent the "state of the art" in quantum computing[1]. They are QLA [42], CQLA [66], and our work, Qalypso [34], and can be viewed as a spectrum from inflexible to flexible ancilla distribution. They differ in their configuration of compute regions, ancilla generation areas, memory regions (for idle qubits), and teleportation network resources (for longer-distance communication)[33, 42].

The QLA architecture is most like a classical FPGA, in that all elements are identical: each element contains enough resources to perform a two-bit quantum gate. Each such *compute region* contains dedicated ancilla generation resources, space for two encoded quantum bits, and a dedicated teleportation router for communication.

CQLA improves upon QLA by allowing two different types of data regions: *compute regions* (identical to those in QLA) and *memory regions* (which store eight quantum bits) [66]. To account for different failure modes (idle errors vs interaction errors), data in memory regions are encoded differently from data in compute regions.

Finally, Qalypso improves upon CQLA by further relaxing the strict assignment of ancilla generation resources. It allows optimized, pipelined ancilla generators to feed regions of data bits (compute regions) that can perform more than just two-bit gates.

[1]Since circuits are *mapped* to these datapaths, they are not quite "architectures" but rather raw material for constructing architectures.

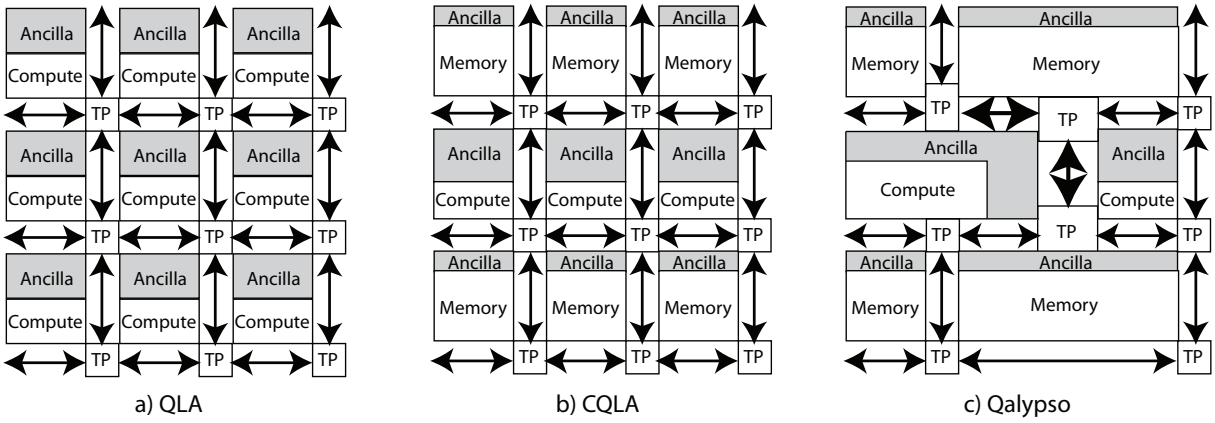



Figure 3.5: Quantum Datapath Organizations: a) Quantum Logic Array (*QLA*): An FPGA-style sea of quantum two-bit gates (compute tiles), where each gate has dedicated ancilla resources. b) Compressed QLA (*CQLA*): QLA compute tiles surrounded by denser memory tiles. c) *Qalypso*: Variable sized compute and memory tiles with shared ancilla resources for each tile; teleportation network can have variable bandwidth links.

The sizing of ancilla generators and data regions can be customized based on circuit requirements. Qalypso requires analysis (Section 3.6) to balance ancilla consumption with ancilla generation. Such analysis can automatically adjust the amount of ancilla bandwidth required in memory regions based on the residency time of qubits.

In all three organizations, each compute or memory region is placed adjacent to a teleport router. Qubits are moved ballistically within regions and teleported between regions.

Proper design of the datapath elements (such as teleportation routers or ancilla generators) is an important factor. In all these microarchitectures, we also pay careful attention to the teleportation network [33, 42]. We have produced layouts for the routers and EPR generators and utilize these in computing area, latency, and error probability of circuits.

In our prior work [34, 68], we study the major datapath organizations in Figure 3.5 and determine that our Qalypso architecture outperforms the other structures. Consequently we will focus the remainder of this work on studying Qalypso based datapaths.

### 3.4.2 Network Synthesis

For large quantum circuits a teleportation based communication network must be inserted into the datapath to perform the long-distance moves. The network synthesis phase inserts the necessary network components into the datapath as further described in Chapter 5.

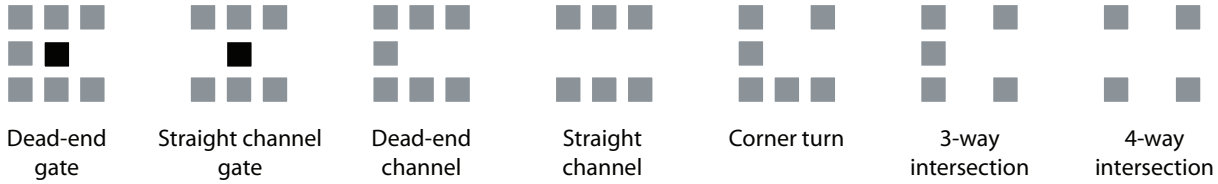



Figure 3.6: Library of ion trap macroblocks. Gray boxes represent electrodes and the black box represents a trap region capable of performing a gate operation. Gates are not allowed in the intersection or turn macroblocks as these trap regions are not as stable as a trap region between two electrodes.

## 3.5 Ion Trap Layout

The currently version of the CAD flow is designed to study quantum datapaths based on ion-trap technology. As mentioned in Section 2.5 there are a number of different experimental implementations of quantum computing using ion-trap technology. Since details of the varying implementations differ in laser positioning and electrode shape and spacing we created the *macroblock* abstraction and ultimately specify all of our datapaths in terms of macroblocks.

The library of macroblocks we use to create our quantum datapath is shown in Figure 3.6. The idea is that while many of the details of a ion trap will change, we will still most likely have designated positions for gates, and channels for moving qubits around with 90 degree turns. Since our heuristics use this abstracted view of communication and gate blocks, we could use these techniques on other technologies as well.

The next step is to take our optimized circuit and create a layout of physical components. Not only does this layout give us a design that can then be fabricated, but it also allows us to create a precise schedule of qubit movement and qubit idling in addition to gates. We go into detail on how this scheduling is done in Chapter 4. The schedule of all the qubits' operations can then be extracted to an error model that accounts for all error sources.

### 3.5.1 Layout Graph Representation

Our layouts are represented by a layout graph which contains macroblock nodes (as mentioned in Section 4.1) that are linked together with QNets. The QNets hold information on how connected macroblocks are oriented with respect to one another. Figure 3.7 shows an example of a layout graph structure. Macroblock nodes specify their location and orientation on the substrate. They also contain additional information to be used by the scheduler to track ion movement through the macroblocks.

Layout graphs can have a similar modular structure as our dataflow graphs have. An abstract layout module can refer to a single macroblock or another layout graph. The embedding of a complex layout module is not as simple as in the dataflow case since the sublayout must be spatially fit into the higher-level design, but layout modularity again gives us considerable savings in representing the full layout since many structures

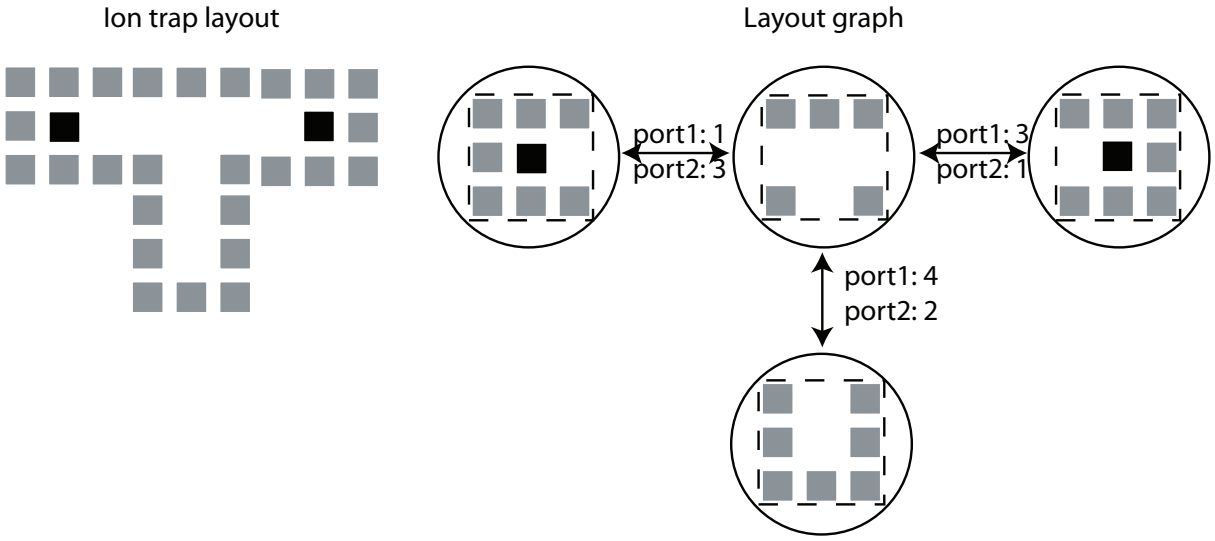



Figure 3.7: A layout and its associated graph. The nodes correspond to macroblocks and the edges correspond to "qnets" which do not have any associated physical entity but determine how macroblocks are oriented with respect to each other.

are often repeated. Some examples of repeated sublayouts are teleportation routers and ancilla factories.

### 3.5.2 Modular Layouts

Our layout specification consists of a sequence of layout module instances, all parametrized by location and a rotation angle, in an XML format. At the lowest level, everything is made up of macroblocks for the underlying technology, like those shown in Figure 3.6. Additionally, we can define higher-level modules, made out of macroblocks, which can then have instances placed in the layout. Higher-level modules must define ports where they connect up to adjacent modules so that the qubit movement scheduler can track movements across module boundaries. Figure 3.8 show an example of such a modular layout.

### 3.5.3 Layout Metrics

Similar to the case of classical CMOS metrics we are also interested in area and delay for layouts of quantum circuits as well. Design area is an important consideration especially in light of the fact that previous work has estimated a design for Shor's factorization of a 1024-bit number to be $0.9m^2$ in area. Since the technology under consideration uses a silicon substrate, fabrication considerations dictate more area-efficiency in our designs.

Delay is another important consideration, since quantum computers promise to perform certain tasks asymptotically faster than classical computers. It is important that the constants involved do not swallow up asymptotic gains.

```
<define_module>
  <type>horseshoe</type>
  <module>
    <type>straight_channel</type>
    <location>0,0</location>
    <rotation>0</rotation>
  </module>
  <module>
    <type>turn</type>
    <location>30,0</location>
    <rotation>0</rotation>
  </module>
  <module>
    <type>turn</type>
    <location>30,30</location>
    <rotation>90</rotation>
  </module>
  <module>
    <type>straight_channel</type>
    <location>0,30</location>
    <rotation>0</rotation>
  </module>
</define_module>
```

```
<module>
  <type>horseshoe</type>
  <location>0,0</location>
  <rotation>0</rotation>
</module>
<module>
  <type>horseshoe</type>
  <location>60,0</location>
  <rotation>180</rotation>
</module>
```

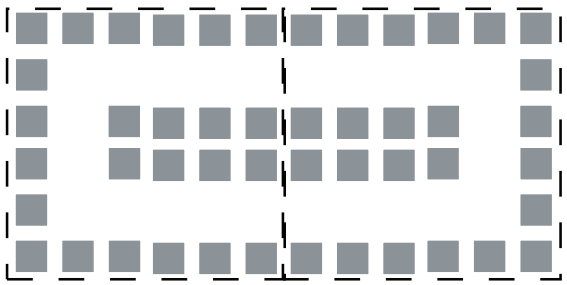


Figure 3.8: Layouts can consist of placements of single macroblocks or definition and then instantiation of larger layout blocks. In this example, we define a larger "horseshoe" block made up of macroblocks and then instantiate two of them in different positions and orientations.

## 3.6 High-level Mapping

Given a tiled datapath such as Qalypso, we must have a method to map groups of circuit elements from our application onto various compute regions in the datapath. The mapper determines where each data qubit will reside during the course of the execution as well as when and where each quantum gate will execute. It starts with a coarse-grained partitioning of gates to compute-regions that minimizes communication. Next, the mapper attempts to schedule each gate operation so that it occurs as late as possible, while prioritizing operations on the critical path. As the mapper progresses, it tracks the location and times of all gate operations, error corrections, and network connections needed to perform the quantum circuit. The mapper discourages imbalanced mappings, such as those that over utilize network links or ancilla generation resources. If the target datapath has fixed ancilla generation resources, the mapper attempts to map operations to regions with unused ancilla bandwidth. In datapaths with flexible ancilla generation,

like Qalypso, the mapper assumes that operations will never wait for ancilla, while still attempting to balance ancilla usage. A later phase matches ancilla generation resources to demand.

## 3.7 Network Routing and Scheduling

Given a mapping of gates to compute regions, we move on to the network routing and scheduling phase where all the long-distance communication within the network is handled. The high-level mapping mentioned in Section 3.6 creates a list of all the long-distance communication that is necessary to run the circuit. The routing phase creates a full route through the network for each of these communication connections. Additionally, the routing phase is responsible for optimizing the connection scheduling to minimize overall circuit latency. The details of this process are described in Chapter 6.

## 3.8 Low-level Scheduling

The only way to properly determine the latency of the application circuit is to schedule the low-level ion trap movement operations in the datapath. This stage of the tools uses the circuit graph to schedule gates and movement at the macroblock level. The details of how scheduling is accomplished are described in Chapter 4.

## 3.9 Fault Tolerance Verification

After a circuit has been mapped to one of the datapaths described in Section 3.4, we must evaluate the resulting error behavior. The goal of the fault tolerance verification tool is to determine the probability of an unrecoverable error on the qubits that would yield an incorrect answer to our computation. Furthermore, we would like to know at which points in the design are data most likely to incur errors.

A circuit is considered to have failed if:

- The circuit is not encoded in an error correction code and one of the output data qubits incurs an error.
- The circuit is encoded and an encoded output qubit incurs more errors than the code can correct.

We classify errors into three categories [3, 65]:

- **Gate:** Errors that occur while manipulating quantum data. This category is the most widely studied in association with error correction performance. It is generally believed that gate errors will be the most significant of the three types.

- **Movement:** Errors that result from moving quantum data. For example, ballistic ion movement in trapped ion technology involves accelerating and decelerating ions both linearly and around corners[30]. This movement induces unwanted vibrations or even collisions, disrupting the internal state that represents the datum.

- **Idle:** Errors in idle qubits that result from spontaneous quantum effects. It is generally thought that idle or storage errors are the least severe of the three per unit time for most technologies.

In principle, we must simulate a circuit from start to finish, injecting errors by assuming that every gate, qubit movement, and qubit stall has an associated error rate and that every error event injects either a bit flip, phase flip or both. Unfortunately, interesting circuits are too large to do simulate exactly—leading to a need for a hybrid methodology.

**Hybrid Error Modeling:** Large designs are specified hierarchically, as a tree of modules. While we can synthesize a complete macroblock layout with fine-grained placement and routing for smaller modules, high-level modules are better handled via coarse-grained mapping techniques. Our mapper does not create exact macroblock specifications for all inter-block channels but instead relies on estimates of ballistic movement and teleportation based on inter-block distances. Further, the distance traveled from ancilla factories to data bits is estimated (quite accurately) after data bits have been placed. Consequently, we utilize a hybrid simulation model in computing communication costs and qubit idle times: not every qubit movement is simulated, but rather aggregate movements are computed and combined to speed simulation and support hierarchical design.

The calculation of the error probability for a mixed ballistic movement model involves three types of information: exact error probabilities, errors from estimated ballistic channels, and errors from teleportation channels. For smaller, leaf modules, we extract error properties exactly through simulation. The coarse-mapped distance estimates for longer ballistic communications are translated into a count of straight and turn macroblocks traversed, yielding error fidelity numbers for traversing these channels.

Finally, the effects of teleportation are determined by computing the fidelity and bandwidth of EPR bits in the channel. We have a model of EPR generation, routing, and purification which permits accurate computation of the latency to setup a teleportation channel as well as the fidelity of the EPR bits available for it. We compute the gate and movement errors within routers along the path, EPR generators, and purifiers.

Consequently, to compute the error probability of a large, hierarchically specified circuit, we combine inter-block movement and idle errors from ballistic and teleportation channels with the exact gate, movement, and idle errors in the compute and ancilla regions to produce one sequential list of possible error points in the layout as the circuit executes. This error list is passed to the Monte Carlo error simulator.

**Monte Carlo Error Simulation:** To propagate errors, we utilize Monte Carlo (MC) simulation [62, 17], in which errors are sampled for each circuit element and errors are propagated accordingly. We traverse a graph representing the full circuit and layout,

sampling gate, movement and idle errors (some of which are aggregate error counts for ballistic and teleportation-based communication) on each qubit in dataflow order. If the final state of the qubits results in an uncorrectable error, the run is counted as a failure. This process is repeated many times to get a statistically significant sample. Our tool uses the Colt JET library [12] version of the Mersenne Twister random number generator.

## 3.10 Benchmarks and Evaluation

Our CAD flow allows us to evaluate arbitrary application circuits. Any circuit that can be specified in QASM can by synthesized and laid out by our tools to generate a detailed layout and schedule of operations. Because of its wide spread interest, we concentrate most of our study on the components of Shor's Factorization algorithm. However, we are also interested in how our approach performs on other quantum circuits. To augment our study, we generate random circuits with varying communication patterns and use these as inputs into our CAD flow.

### 3.10.1 Adder Circuits and Shor's Algorithm

Since Shor's Factorization algorithm generates significant interest, we use it as a major benchmark within our study. Chapter 7 describes the algorithm in more detail along with the various components that compose it, one of which is the Quantum Adder circuit. We study two different structures of quantum adders: a ripple-carry adder and a carry look-ahead adder and use these modules to study a full implementation of Shor's Algorithm.

### 3.10.2 Random Circuit Generation

In addition to the real application benchmarks we mentioned, it is convenient to have benchmark circuits in which we can exert more control over various properties, such as the number of qubits, gates, or overall communication structure. For this reason, we developed a method for synthesizing random quantum circuits to test various portions of our tool flow. The generated random circuits have the following parameters:

**Gate count:** Number of total gates that are in this circuit.

**Gate type:** Types of gates included in the random circuit. Typically, we focus on the gates that appear most often in our applications, CNOT, Hadamard, and some non-transversal gate like T are common choices.

**Qubit count:** Number of data qubits that are operated upon in the circuit.

**Splitting fraction:** The splitting fraction tells us how to group gates when we are determining what gates should connect to each other when generating the circuit. A fraction of 0.5 will generate a circuit by successively breaking it into 2 equal sized

parts and adding connections within each part, then recursively dividing each subpart in half. A fraction of 0.9 will divide the circuit into one with 10% of the gates and another with 90% of the gates and follow the same recursive procedure.

### 3.10.3 ADCR: An Aggregate Metric for Probabilistic Computation

As mentioned earlier, delay and success probability are closely connected in the evaluation of any quantum circuit. We are not simply interested in a single run of a circuit on a layout if it does not produce the correct answer. We would instead like to know the expected time to get a correct answer:

$$E(Delay) = Delay_{singlerun} \times E(runs\ for\ correct\ result) \tag{3.1}$$

$$= Delay_{singlerun} \times \sum_{n=1}^{\infty} \frac{n}{P_{success}(1 - P_{success})^{n-1}} \tag{3.2}$$

$$= Delay_{single_run} \times \frac{1}{P_{success}} \tag{3.3}$$

Prior work has focused on maximizing the overall success probability $P_{success}$ at all costs. This goal might be a suitable sacrifice if we are always on the verge of a catastrophic decline in success probability for a design (probably the case in all current laboratory setups). As the technology matures, it will be more important to evaluate all the design considerations; for example, reducing a layout's delay by 10x with only a 10% reduction in success probability may be a worthwhile trade-off to consider. Critical to this evaluation is a comprehensive evaluation of the overall probability of success of a design. If we overestimate this probability, we could end up making trade-offs resulting in a design that does not work at all.

To evaluate the quality of quantum layouts with a single number, we propose a *composite* metric called *Area-Delay-to-Correct-Result (ADCR)*. ADCR is the probabilistic equivalent of the Area-Delay product from classical circuit evaluations:

$$\text{ADCR} = \text{Area} \times E(\text{Latency}_{\text{total}}) \tag{3.4}$$

$$= \text{Area} \times \sum_{n=1}^{\infty} n \cdot \text{Latency}_{\text{single}} \cdot P_{\text{success}}(1 - P_{\text{success}})^{n-1} \tag{3.5}$$

$$= \text{Area} \times \frac{\text{Latency}_{\text{single}}}{P_{\text{success}}} \tag{3.6}$$

For ADCR, *lower is better*. By incorporating potential for circuit failure, ADCR provides a useful metric to evaluate the area efficiency of probabilistic circuits. It highlights, for instance, layouts that use less area for the same latency and success probability. Or, layouts that use the same area for lower latency or higher success probability.

### ADCR-optimal

With the definition of ADCR, we can now talk about designs that are *ADCR-optimal*, or the set of design parameters the yields the lowest ADCR. Since ADCR is meant to be a comprehensive metric for all the stages in our CAD flow, finding the design with the best ADCR takes some amount of iteration and feedback.

In order to obtain an ADCR-optimal layout for an application circuit we must search over these parameters:

**Error correcting code selection:** Different codes impact area, latency and success probability, due to encoder size, complexity and errors corrected. Thus we must iterate through different encodings in the QEC synthesis phase, optimizing, laying out the encoded circuit, and simulate for failures.

**QEC optimization level:** We can tune the degree of QEC optimization in order to trade-off success probability for area and/or latency. Thus, we must iterate over different optimization settings, lay out the resulting circuit, and simulate for failures to find a design with a good ADCR value.

**Datapath configuration:** Within the datapath we can vary a number of parameters ranging from the compute datapath to the network structure. Varying the number and size of compute regions or number of communication links in the network impacts area, delay and thus we must iterate through different designs in the datapath space, then lay out and simulate for failures.

All these choices leads to a multi-parameter optimization problem. We currently binary search through many of these parameter spaces to find ADCR-optimal designs, but future work includes using better heuristics to narrow the set of candidate parameter settings to converge on a good design faster.

# Chapter 4

# Optimizing Short-distance Quantum Communication

At the lowest level, the quantum datapaths we study are constructed from ion-traps. A quantum circuit specified as a series of quantum gates (such as those described in Section 3.1), must be mapped into a series of operations within this datapath to ultimately execute the desired circuit. A large part of what occurs in the datapath is centered around short-distance communication needs: moving qubits from one trap location to another in order to perform gate operations.

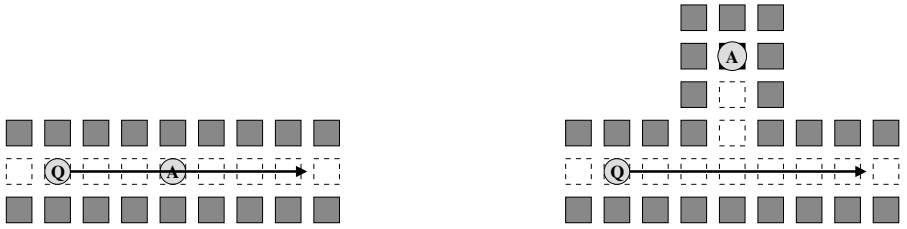
As mentioned in Section 2.5.6, ion-trap datapaths take on a planar structure and consist of a number of electrodes and trapping regions. Moving a qubit through this type of datapath requires carefully manipulating the electrodes to push and pull ions into and out of trap regions. Additionally, we must be mindful of qubit locations within the datapath as we cannot allow qubits to collide with other qubits as they move throughout the datapath.

## 4.1 Ion Trap Datapath

To construct a datapath for a given quantum circuit we first analyze the circuit to determine it's computational and communication needs. For each gate in the circuit, we must find a location within the datapath that is capable of performing that gate. Unlike in a classical circuit, each gate location in our datapaths can be time-multiplexed to perform multiple gates. However, we must be careful. With too few gate locations, a datapath can adversely effect overall circuit latency through gate serialization.

Qubits must be physically adjacent to each other to perform a double-qubit operation such as a `CNOT`. The gate locations in the datapath are interconnected with ballistic movement communication channels, the quantum equivalent of classical wires. Moving a qubit from gate location to another is accomplished by using movement protocols as described in Section 2.5.2. Qubits move through a series of ion traps until the destination trap is reached.

While two qubits can reside in the same ion trap to perform gate operations, a qubit




(a) Invalid movement operation.

(b) Valid movement operation.

Figure 4.1: Example of invalid and valid movement operations. The operation in (a) is not allowed since qubit Q cannot move through the trap occupied by qubit A. The operation in (b) is allowed as qubit A is now far enough away from the path qubit Q wants to take.

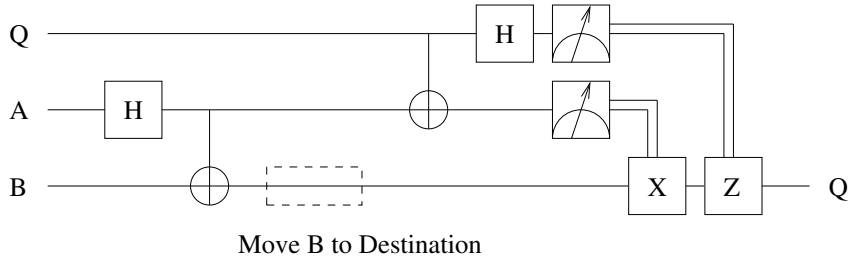



Figure 4.2: Circuit used to teleport qubit Q to some destination. Initially qubits A and B are entangled to form an EPR pair. Qubit B is then moved to the destination location. After interacting qubits Q and A, their states are measured and the results are classically transmitted to the destination. Using this classical information B is operated on to recreate Q at the destination.

cannot travel through another qubit. Therefore, all the traps a qubit moves through to reach a destination must not be occupied by other qubits as illustrated in Figure 4.1. Figure 4.1(a) shows an invalid movement operation. Qubit Q cannot take a path that goes through the trap occupied by qubit A. The path shown in Figure 4.1(b) shows a valid movement as now qubit Q is traveling along a clear path. To prevent these types of collisions from occurring the datapath must be carefully designed to provide enough communication channels.

As an example of a circuit-specific datapath, we will consider an implementation of the Teleportation circuit, shown in Figure 4.2. This circuit consists of three qubits Q, A, and B and uses the quantum circuit notation introduced in Section 2.2.

Recall from Section 2.4.1 that teleporting a data bit is performed by entangling two qubits to form an EPR pair (shown in the figure as qubits A and B), moving one of the qubits to the destination location (qubit B), interacting the data bit and the remaining EPR pair qubit (qubits Q and A), measuring their values, and using the measurement results to perform operations on the destination EPR pair qubit (qubit B) to recreate the data qubit.

One possible implementation of the quantum datapath for this circuit is presented

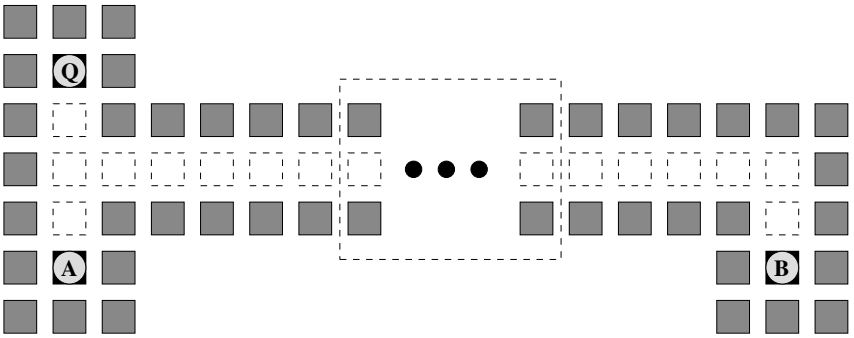



Figure 4.3: One possible datapath implementation for the teleportation circuit shown in 4.2. Qubit B is shown in the destination ion trap to where qubit Q will ultimately be teleported. The large dashed box in the middle represents an arbitrary distance.

in Figure 4.3. The left side of the figure is where the initial EPR pair generation takes place. After the EPR pair is created, qubit B moves to the destination location. The large dashed rectangle represents an arbitrary distance qubit B moves to eventually get to the destination. With the datapath in this state, qubits Q and A are now able to interact to perform the teleportation operation.

In ion trap technology, given unlimited resources, it is possible to create a datapath where every trap is capable of performing a gate operation. However, assuming unlimited laser resources is unrealistic and therefore we limit the number of ion traps capable of performing gate operations and always explicitly mark them. The teleport datapath shown in Figure 4.3 only contains three ion traps with gate capabilities. The rest of the traps are used for movement.

## 4.2 Datapath Control

The quantum datapath is useless without any classical control. The classical control is designed to manipulate the quantum datapath in order to perform the necessary operations. The classical control system manages the ion trap electrodes, all laser resources used to perform gate operations, and the measurement device.

We begin by discussing the control requirements for the ion traps. As described in Section 2.5.1, each ion trap has some number of electrodes that must be individually controlled to effectively confine an ion. We move ions between traps by applying voltage sequences to the relevant electrodes, and then stabilize the electrode voltages to confine the ion within the trap. Regardless of which ion-trap implementation is used, all changes to the electrode voltages must be done in small increments and with precise timing to minimize the amount of energy gained by the ion. Currently, experimental demonstrations of ion-trap technology utilize Digital-to-Analog Converters (DACs) to accomplish this task [50, 32, 70]. The DACs allow designers to specify a sequence of discrete voltage levels which are then converted into the intricate voltage sequences required for the various trap operations.

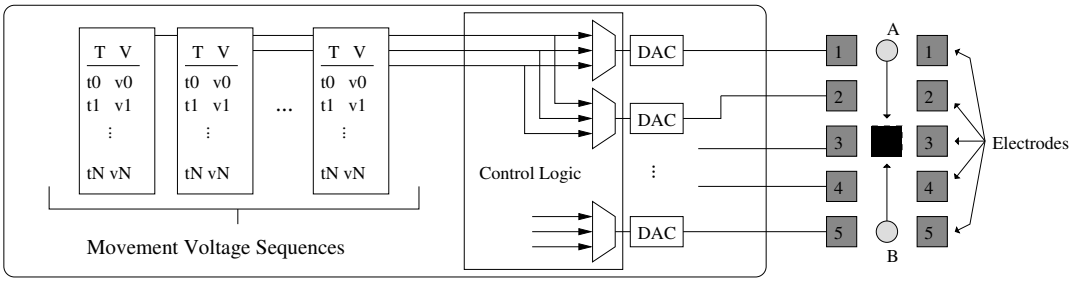



Figure 4.4: DACs are used to control the voltages of the ion trap electrodes. Electrodes that share a label receive the same voltage sequence. The control logic programs the DACs to generate the necessary voltage sequences. The voltage sequences corresponding to the various movement operations are stored in a memory.

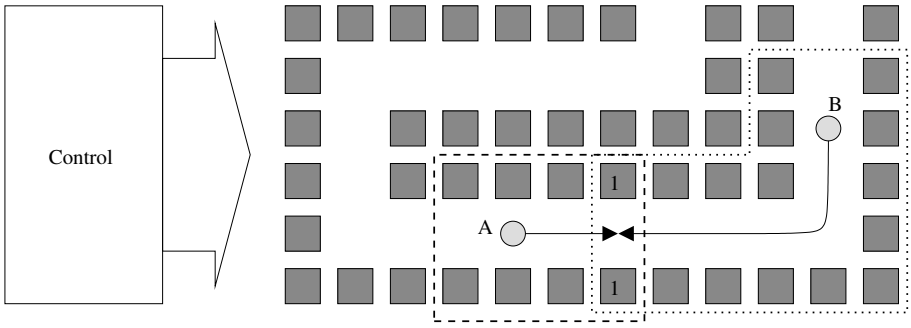



Figure 4.5: Electrodes required to move two qubits to the trap region between the electrodes labeled 1. The movement of qubit A requires all the electrodes within the dashed rectangle. The movement of qubit B requires all the electrodes within the region enclosed by the dotted line.

A method to integrate DACs into the control system is outlined in Figure 4.4. On the left are a set of DACs fed by various movement voltage sequences. Each primitive operation has a defined voltage sequence, essentially a list of time steps with the corresponding voltage levels for the electrodes, stored in RAM. Multiplexers are set by the control system to feed the DACs with the appropriate voltage sequence for the desired movement operation. The control logic is responsible for deciding where qubits need to go and in what order to perform the operations. Once this decision is made, the control logic retrieves the predetermined voltage sequences for the desired operation and uses this information to program the DACs corresponding to the electrodes that are participating in the operation. In this example, qubit A and qubit B need to perform a `CNOT` operation. To do so, they must be co-located in the trap between the electrodes labeled 3, so the control logic programs the DACs to move them from trap to trap until they reach their destination. For qubit A, first electrode groups 1 and 2 are set to perform a single trap to trap movement from 1 to 2. Then the same operation is again performed, but this time using electrode groups 2 and 3. The same operations are performed to move qubit B.

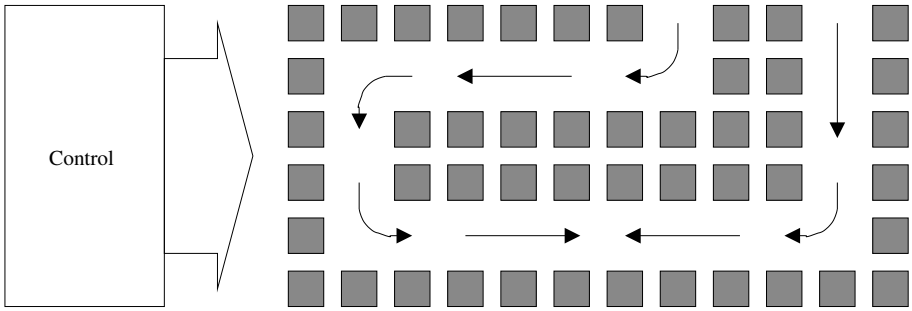



Figure 4.6: The same datapath shown in Figure 4.5 redrawn to highlight repetitive operations that can be exploited by a hierarchical control system. The control logic required to perform the straight arrow movement is the same in all cases. Similarly, the control to perform the turn arrows is also the same in each turn case.

Performing these types of movement operations requires the coordination of multiple adjacent trap electrodes as ions cross from one trap into another. For example, in Figure 4.5, we see two qubits that will be moved from their current location to the trap between the electrodes labeled 1. All the electrodes contained within the dashed rectangle are required to move qubit A, and all the electrodes surrounded by the dotted line are required to move qubit B. The control system must individually control each of these electrodes to allow these simultaneous movements to occur.

### 4.2.1 Macroblock Abstraction

Designing a control system to track and manipulate a few ion traps and qubits is not terribly challenging. As the datapath grows in size, however, this design problem becomes increasingly complex, as more electrodes must be controlled and additional qubits must be tracked. Moreover, even after a control system is designed for a specific datapath, any changes to this datapath requires modifications to the control system to accommodate these changes. Some of these issues can be addressed by utilizing a hierarchical control system design that isolates changes in low-level control from higher-level control logic.

If we look at Figure 4.6 (the same datapath shown in Figure 4.5) then we can see there are a number of repeated operations that occur. All the straight arrow movements utilize the exact same control logic. The same holds true for all the turn arrows. By analyzing a number of datapaths, we can identify a set of repetitive control actions that can be turned into a library of *macroblocks*.

Our macroblocks are simple abstractions that combine multiple trap regions and electrodes into a single group. The macroblocks can be designed to include any number of electrodes and trap regions. Each macroblock has an associated control unit to handle all its internal control and provide interface logic to other macroblock control units and a higher level controller. We purposely design our macroblocks so they only interact with surrounding macroblocks when a qubit needs to travel between them. In all other cases the macroblock need only be concerned with its internal state.

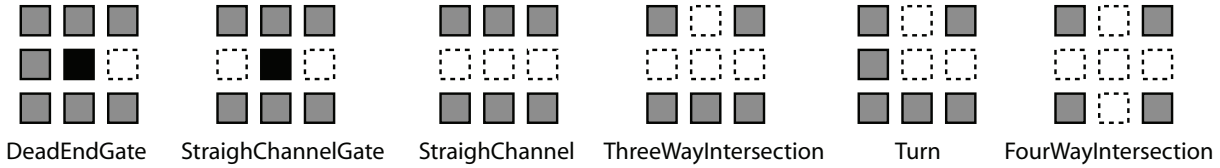



Figure 4.7: Example library of ion trap macroblocks. Gray boxes represent electrodes and the black box represents a trap region capable of performing a gate operation. Gates are not allowed in the intersection or turn macroblocks as these trap regions are not as stable as a trap region between two electrodes.

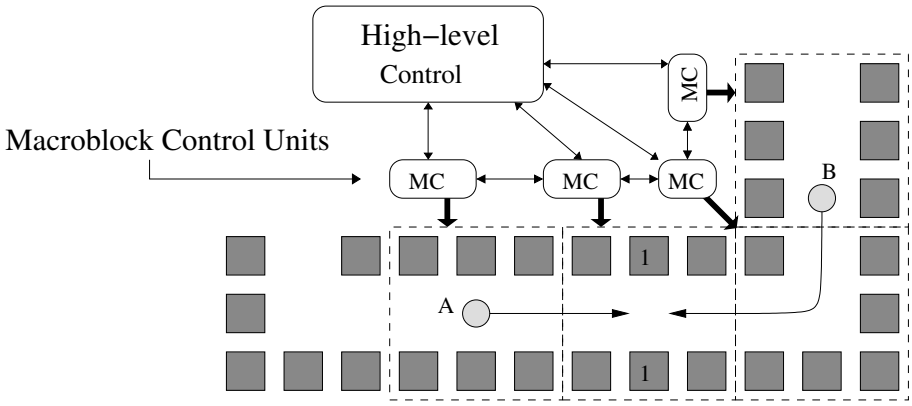



Figure 4.8: The same movement as pictured in Figure 4.5, but with the macroblock abstractions inserted. The macroblock control units are shown along with their interface to high-level control.

The macroblocks we use in our datapath evaluations are shown in Figure 4.7. They consist of four communication macroblocks: StraightChannel, ThreeWayIntersection, Turn, and FourWayIntersecion, and two gate operation capable macroblocks: DeadEndGate and StraightChannelGate. We do not permit gate operations inside the intersection or turn macroblocks as the trap regions within them are unlikely to be stable enough. If, in the future, this assumption proves to be untrue, adding new macroblocks into our library would be a simple task.

All of our macroblocks are sized identically, but designing macroblocks of varying sizes is possible. We choose to use uniform sizes because it greatly simplifies the creation of datapaths by allowing us to use simple grids for layout. Additionally, all openings in the macroblocks are always centered on the side allowing us to place macroblocks next to any other macroblock with a corresponding opening without complex alignment procedures.

To illustrate how macroblocks are used, we build the example datapath shown in Figure 4.5 using our macroblocks and show the result in Figure 4.8. Rather than a single control block required to manage all the electrodes we create a modular system where smaller control units are responsible for fewer electrodes thereby minimizing complexity. Placing the macroblocks next to each other automatically connects the lower-level control units together. All that remains is designing the high-level control.

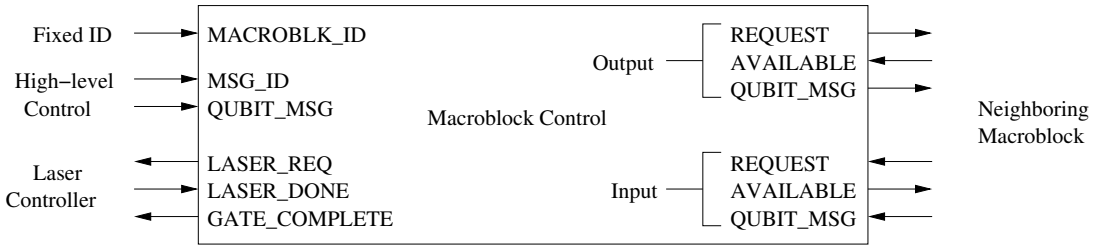



Figure 4.9: Macroblock control interface. For each quantum port in the macroblock, there are a set of Input and Output signals that connect to the neighboring macroblock. The macroblock interfaces to higher level controllers through the Instruction Controller and Laser Controller interfaces. Each macroblock also has a unique ID for addressing purposes.

### 4.2.2 Macroblock Sequencing

The interface used by our macroblocks is shown in Figure 4.9. Each macroblock interfaces with high-level control and the laser control. Additionally, macroblocks have a set of input and output signals for each datapath port it contains, shown on the right side of the figure. These signals connect the macroblock control to the control unit of the neighboring macroblock(s). The `REQUEST` and `AVAILABLE` signals are used to determine when qubit movement is permitted. If a macroblock containing a qubit is instructed to move the qubit to a neighboring macroblock it first asserts the `REQUEST` signal associated with the port out of which the qubit will travel. The neighboring macroblock then asserts `AVAILABLE` if the qubit movement is permitted and on the following cycle the qubit moves into the new macroblock along with a qubit command message (described below). This simple handshaking prevents qubits from entering macroblocks that already contain qubits and prevents qubit collisions from occurring. Only macroblocks that support gate operations (DeadEndGate and StraightChannelGate) allow two qubits to reside within them simultaneously.

Each macroblock is capable of receiving a qubit control message. Using these qubit control messages, macroblock control logic blocks can determine where to move qubits and when to execute a gate operation. Qubit control messages are simple bit streams composed of a qubit ID along with a sequence of commands. When a qubit needs to perform an action the high-level control logic sends the macroblock a control message. This control message then travels with the qubit as it traverses the datapath. Once a macroblock receives a qubit and its corresponding control message, it uses the first command in the sequence to determine the operation it must perform. The macroblock then removes the command bits it used and passes on the remaining control message to the next macroblock into which the qubit travels. In this manner, the high-level control logic can create a multi-command qubit control message that specifies the path a qubit will traverse through consecutive macroblocks along with where gate operations take place. The high-level control logic only has to transmit this control message to the source macroblock, relying on the inter-macroblock communication interface to handle the rest.

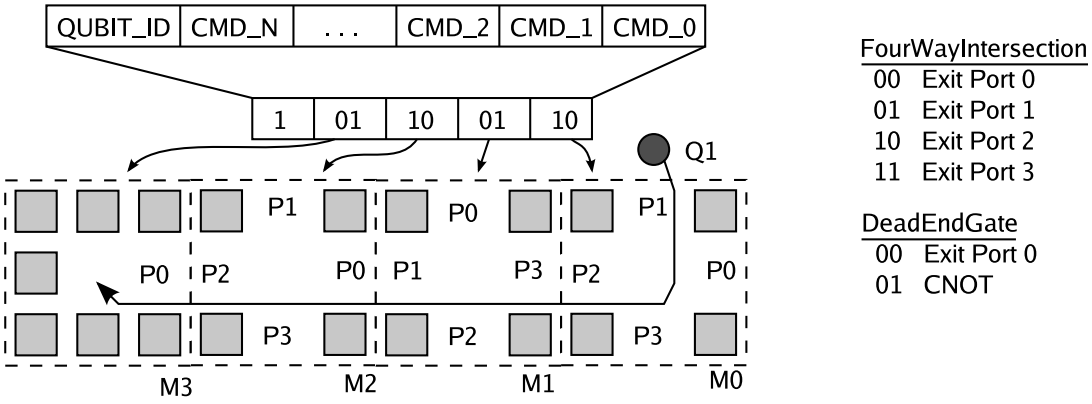



Figure 4.10: Example of how a qubit control message is constructed to move a qubit through a series of macroblocks. The layout is constructed from four macroblocks. In this example, M1 happens to be rotated 90 degrees compared to M2 and M0. The qubit enters M0 and travels through M1 and M2, arriving at M3 where it is instructed to perform a CNOT.

Figure 4.10 illustrates how macroblocks accept control messages from the high-level control logic and how the messages travel along the path. In this example, the high-level control determined the path Q1 will take: M0→M1→M2→M3. This path is converted into a single control message as shown in the figure and sent to M0. M0 looks at the first two bits of the control message and determines the qubit must exit port P2. After conducting the handshaking protocol with M1 as described above, Q1 moves from M0 to M1 and the command message is passed to M1's control unit with the lowest two bits removed. M1 now looks at the low bits of the command message to determine that Q1 must exit through port P1. This process repeats until the qubit reaches M3 where the command message indicates a gate operation will take place. Once the gate operation completes (as indicated by the `GATE_COMPLETE` signal) the high-level control will issue a new command message to M3 to detail the next path Q1 will take. If no command message is received, the qubit remains in M3 until the new message arrives.

Communication between the high-level control logic and the macroblocks takes place using a shared control message bus to minimize the number of wire connections required. Each macroblock listens to the control message bus for messages addressed to it and only processes messages with destination IDs that match the macroblock's ID. A macroblock is only responsible for monitoring the control message bus if it contains a qubit that has no remaining command bits. This condition generally occurs after a gate operation takes place when the high-level control logic is deciding what action the qubit should take next. Once a new control message is received for a qubit, the macroblock resumes operation.

The macroblock interface enables the high-level control logic to schedule qubit movement as a path through a sequence of macroblocks without concerning itself with the low level details of qubit movement. This modular system allows macroblocks to be replaced with any other macroblock that implements the defined interface with minimal changes to the high-level control logic. The high-level control logic must always know the datapath

structure as it needs this information to determine qubit paths, however, the high-level control is insulated from changes that may occur in the macroblock control units such as modifications to the electrode voltage sequences.

### 4.2.3 Lasers and Measurement

As described in Section 2.5.4, performing gate and measurement operations in ion trap technology requires lasers. Lasers are a global resource shared among all macroblocks in the system. When a macroblock is instructed to perform a physical gate operation, it passes a laser request message to its higher-level control. The high-level control combines all laser requests it receives and forwards the request out to the laser control unit. The laser control unit aggregates incoming laser requests and fires off the desired pulses at appropriately synchronized times. Because of the number of simultaneous requests that will occur at every time step and what will most likely be a limited number of available lasers, the laser control unit must act as an arbitration unit. It aggregates all incoming laser requests and decides which requests to grant. Any requests that are not granted at a given time step must stall until a laser resource becomes available. The laser control also manages the laser distribution scheme used and ensures that laser pulses are directed only at the desired macroblocks.

Measurement control operates in a similar fashion. Measurement not only requires a laser resource but also uses a CCD to detect the measured values. All measurements in the system must by synchronized or delayed until the next time unit the CCD becomes available.

### 4.2.4 Interface to High-level Control

The final piece of the control system is the high-level control. This component is responsible for interfacing with all the macroblock control units and the laser and measurement control. The high-level control is designed to accept a set of instructions and translate them into the appropriate macroblock control messages.

With all of these pieces in place, creating a fully functional circuit, given a predefined library of macroblocks, becomes a simple procedure. The datapath is specified using the macroblocks in the library. With this datapath specification, the macroblock control units automatically interface together and with the high-level control to create a functional control system. A series of instructions representing the quantum circuit is then input into the high-level control unit to execute on the datapath. The sequence of instructions details all of the qubit movement requirements and defines when and where gate operations occur. In this manner, arbitrarily complex quantum circuits can be created including a full control system.

### 4.2.5 Scheduling Communication

Qubit movement and communication must be carefully scheduled within a quantum datapath in order to perform the desired circuit. Once we have a datapath built out of

macroblocks we use our scheduler to execute the circuit on the datapath. The scheduler is responsible for deciding the location gates occur, the order in which the gates occur and the movement paths used by the qubits to get from one gate location to another.

Our scheduler tracks each qubit in the datapath individually and determines how they move around the system. It must decide where the next gate operation will occur and allocate channels for the qubits to use to move to this new gate location. Our scheduler is based on a standard priority list scheduling algorithm [27].

The scheduler executes the instruction in dataflow order, prioritizing instructions on the critical path. This method allows the highest priority instruction to run first making it more likely for it to gain access to the required resources. These contested resources include both gates and channels/intersections. Once all possible instructions have been scheduled, time advances until one or more resources are freed and more instructions may be scheduled. This scheduling and stalling cycle continues until the full sequence has been executed or until deadlock is detected, in which case the highest priority unscheduled instruction at the time of deadlock is reported.

It is often the case that we design a datapath with a logical to physical qubit mapping in mind. For example, if we are studying datapath with a $[[7, 1, 3]]$ encoding in mind we know that each logical qubit will be represented by 7 physical qubits. Knowing this mapping, we may choose to allocate additional communication channels to allow all the physical qubits to move in a synchronised manner. The scheduler must be careful to allow this types of moves to occur correctly, otherwise it may not take full advantage of the resources available in the datapath.

Figure 4.11 illustrates an example of a incorrectly scheduled move operation. In this example, the logical qubit is encoded as three physical qubits residing in region A. The goal of the move is to move all three qubits from A to B. A scheduler that only operates at the physical level might schedule each move in series so that they each take the same path, when it is clear that the designer included extra horizontal channels with the intention that all the qubits to move simultaneously as shown in Figure 4.12. In this figure, the three physical qubits are treated as a logical qubit and moved together, optimizing the overall move latency. Our scheduler allows us to annotate logical to physical qubit mappings so that we choose the best move path possible.

## 4.3 Manual Layout of Quantum Circuits

Our CAD tools provide multiple options for creating macroblock layouts. The most hands on and flexible method is to use a graphical layout tool we developed to manually build layouts. Our layout tool is similar in concept to those used in classical CAD flows in that the designer selects modules from a library and manually places them in the layout. A screenshot of the layout tool is shown in Figure 4.13. Users can build layouts using our standard macroblock definitions or they can create more complex modules to add to the library of available modules.

As modules are placed in the layout their respective ports are connected to enable qubit movement between the modules. When the layout is complete, we run design-rule

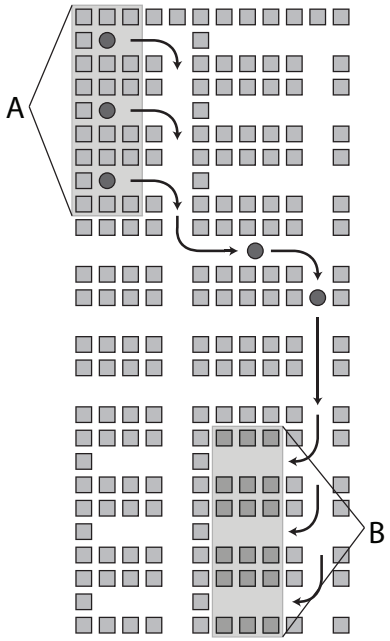



Figure 4.11: A poorly scheduled logical qubit movement. The three qubits at A need to move to B. If each qubit takes the first shortest path they will all follow serially causing congestion at the turn and delaying the overall movement.

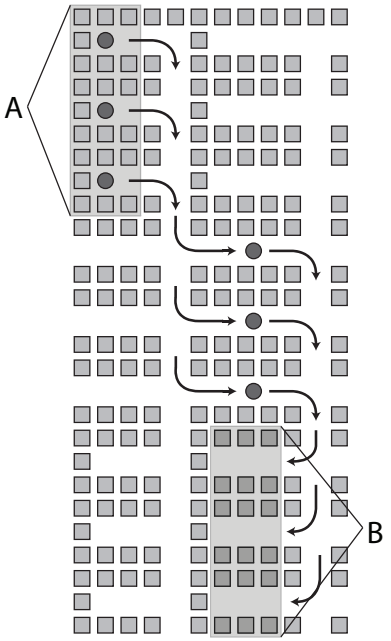



Figure 4.12: A correctly scheduled logical qubit movement. The three qubits moving from A to B do so in parallel reducing congestion at the turns.

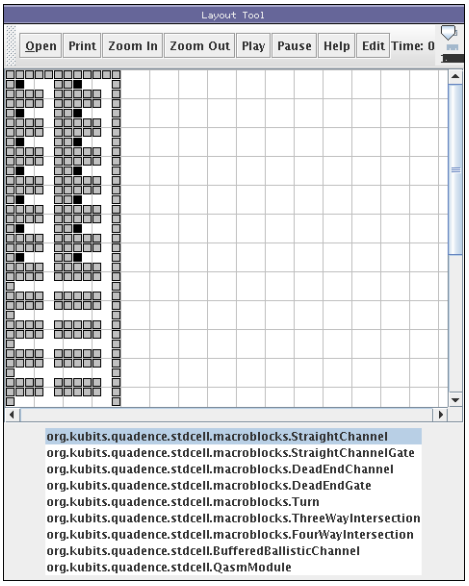



Figure 4.13: Custom designed Layout Tool. The manual layout tool interface allows users to create arbitrary layouts using standard macroblocks or other user-defined modules.

checks to verify the layout is consistent: gate locations cannot be in an unconnected island and channel openings must line up with other channel openings.

The Layout Tool also serves as a way to visualize qubit movement and communication within a datapath. After the CAD tools have processed a circuit (either with a layout manually specified or automatically generated) it outputs a full schedule of operations that occur in the datapath (as described in Section 4.2.5). This information along with the corresponding layout can be loaded into the Layout Tool to observe the full circuit run. Qubit movement is animated through the datapath and qubits are highlighted when gate operations occur allowing visual inspection of the circuit run. This direct observation simplifies the task of spotting congestion issues in smaller layouts.

## 4.4 Automated Grid-Based Layout

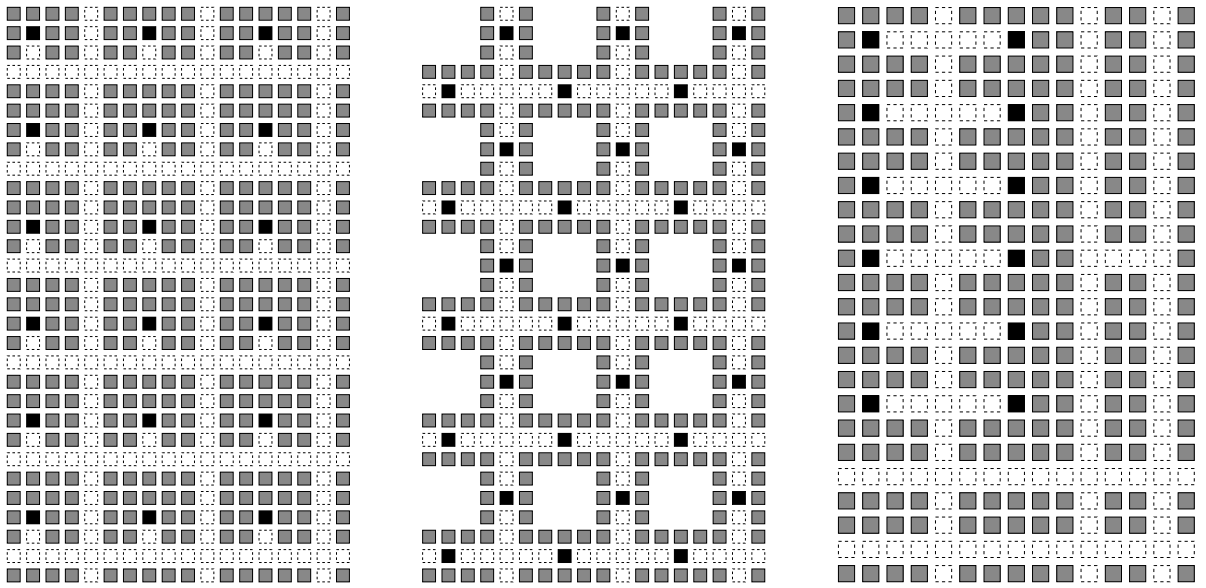
Manually specifying a full circuit layout is convenient for small layouts, however it becomes difficult to use this process to evaluate large quantum circuits. We use specialized automated layout techniques to speedup up the design process and provide assistance to create larger layouts.

One common theme to many of the previously proposed quantum computing datapaths is that they are all created by first constructing a smaller cell and tiling this cell to form a larger layout. We refer to these types of layouts as grid-based layouts, since they are constructed from a grid of primitive cells. In some cases the primitive cell is composed of very few ion traps with only one gate location, in other cases the cell consists of many ion traps with numerous gate locations. Once a primitive cell is designed, it is easy to tile the cell to create an appropriately sized datapath for a given quantum circuit. Furthermore, grid-based structures are very appealing to consider because, apart from selecting the number of cells in the layout and the initial qubit placement, no other customization is required in order to map a quantum circuit onto the layout.

### 4.4.1 Regular Tiled Datapaths

A number of proposals have been made for ion trap datapath designs. Metodi et al [42] initially presented the Quantum Logic Array (QLA) design which used the structure shown in Figure 4.14(a) as its basic building block. The qubits only perform gate operations in the trap regions denoted by the black box. This layout allows qubits to move around while other qubits remain in gate locations as none of the channels can be blocked. Consequently the design requires a large layout to accommodate the communication channels.

Metodi et al [43] additionally propose a layout using the structure shown in 4.14(b). This layout packs in more gate locations per area, but does so at the expense of requiring a more complicated scheduling algorithm as it is possible to trap a qubit by blocking channels. Recall that if a qubit is occupying a trap region, another qubit cannot pass through that trap. Instead, the qubit must move out of the trap region in order to clear a path for the the passing qubit, an unfavorable operation that exposes the qubit to additional sources of error.



(a) QLA Logical Qubit Layout from [42].

(b) QPOS Grid Layout from [43].

(c) Svore/Balensiefer Layout from [64, 3].

Figure 4.14: A sample of physical layouts of Functional Units taken from the literature.

Svore et al [64] and later Balensiefer et al [3] use variations of the design shown in Figure 4.14(c). This design tiles blocks which are composed of regions dedicated to computation surrounded by special communication channels.

While all of these physical layouts have been proposed, very little has been done to determine which layouts are good. Balensiefer et al [3] provide some tools to evaluate the performance of their layouts by varying characteristics such as number of communication channels and gate locations, but do not extend the work to more general structures. In a more recent work, Metodi et al [43] present a Quantum Physical Operations Scheduler (QPOS) that given a quantum circuit description and a layout, can generate a schedule of operations upon that layout, however their QPOS is limited to layouts that conform to the structure shown in Figure 4.14(b).

### 4.4.2 Optimizing Titled Datapaths

We developed a set of tools to create and evaluate these types of grid-based datapaths. Our goal was to exhaustively search through a number of different grid-based datapaths to determine if datapath designs such as the QPOS grid (Figure 4.18) can be improved upon. We limit our search to primitive cells sized $3 \times 3$ and smaller as the amount of compute time to search for larger cells is prohibitive. Once we construct a primitive cell out of macroblocks, we tile the cell to create a larger layout and execute the circuit with this datapath.

An overview of the tools we use for the evaluation is shown in Figure 4.15 and described here:

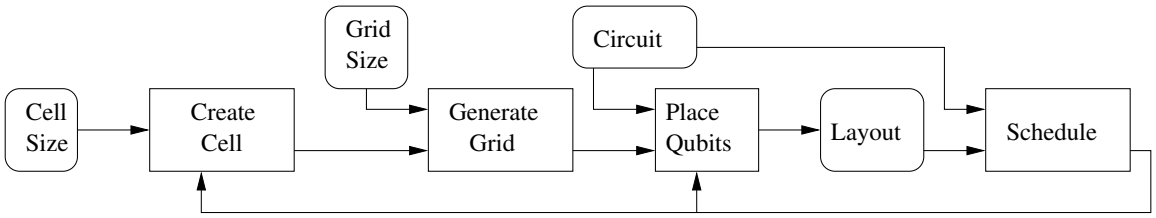



Figure 4.15: Tools used to create and evaluate various grid-based functional unit layouts.

**Create Cell:** The process begins by first deciding on a primitive cell size. We start with the small cells, $1 \times 2$ and $2 \times 1$, and build up to $3 \times 3$ sized cells. Give a cell size, we create the primitive cells out of macroblocks. Each block in the cell can be one of 17 different macroblocks after accounting for rotation (6 of which are shown in Section 4.2.1). The process starts by placing macroblocks in the cell until a valid cell is created. Valid cells must have at least one gate location and at least two open ion traps that when tiled will line up with the neighboring cell's open ion traps. A cell cannot be completely enclosed as when it is tiled qubits will not be able to traverse from one cell into another. Additionally a primitive cell must be internally consistent; an opening from a macroblock must align with the opening of a neighboring macroblock.

**Generate Grid:** Once we have established the primitive cell structure, we move on to generating the full datapath. The datapath is constructed by tiling the primitive cell until we create a grid of the desired size. There is no inherent limit to the maximum size of the grid, but there is a minimum. One of the restrictions we make is that qubits can only "idle" in ion traps that are designated gate regions which only appear in the `StraightChannel` and `DeadEndGate` macroblocks. We make this restriction because traps between two electrodes are more stable than trap regions in intersections. The trap regions in intersections are only used in qubit transit when the qubit is expected to leave the trap immediately after it enters. In an effort to limit the large search space of potential grid sizes, we only use grid dimensions which match the area of the datapaths presented in [43], allowing us to compare the performance of datapaths with matching areas.

**Place Qubits:** Given a full datapath and a quantum circuit to execute, the first step in execution is determining where qubits are initially placed. The two methods we use are: a systematic left to right, one qubit per cell approach, and a randomized placement. The systematic placement allows us to fairly compare different layouts. However, since the initial placement of the qubits can affect the performance of the circuit, the tool also tries a number of random placements in an effort to determine if the systematic placement unfairly handicapped the circuit.

**Schedule:** The scheduler creates a full schedule of movement and gate operations within the datapath and is further described in Section 4.2.5.

This layout generation and evaluation procedure is iterated until all valid cell con-

| Circuit name | Qubit count | Gate count |
|---|---|---|
| $[[7,1,3]]$ L1 encode [60] | 7 | 21 |
| $[[23,1,7]]$ L1 encode [62] | 23 | 116 |
| $[[7,1,3]]$ L2 encode [60] | 49 | 245 |

Table 4.1: List of our QECC benchmarks, with quantum gate count and number of qubits processed in the circuit.

figurations of the given size are searched. We then repeat this process for different cell sizes.

### 4.4.3 Evaluating Grid-Based Datapaths

Using our tool set we evaluated three quantum error correction circuits to determine the effects of datapath structure on runtime. We chose error correction circuits because the high error rates present in all currently demonstrated quantum technologies necessitate the use of some level of error correction to perform all but the simplest tasks. The three circuits we targeted are listed in Table 4.1 and consist of the Steane $[[7,1,3]]$ Code [60] level 1 encode circuit (7 physical bits to represent 1 data bit), the Steane $[[7,1,3]]$ level 2 encode circuit (49 physical bits to 1 data bit) and the Golay $[[23,1,7]]$ Code [62] level 1 encode circuit.

For each circuit we ran our tools to determine the best grid-based datapath structure. We searched grids based on primitive cells sized $2 \times 2$, $2 \times 3$, $3 \times 2$, and $3 \times 3$. As an example, Figure 4.16 shows the results of searching for the best layout composed of $3 \times 2$ sized cells targeting the $[[23,1,7]]$ Golay encode circuit with an area of 143 macroblocks. In this example, we searched over more than 900 valid cell configurations (listed as the X-axis in the graph). For each cell configuration, we try multiple initial qubit placements, resulting in a range of runtimes for each cell configuration (Y-axis). Differences in the runtime of the circuit are not limited to just variations on the cell configuration but are in fact also highly dependent on the initial qubit placement as can be seen by the difference in minimum and maximum times across the different structures. The runtime for the worst structure and initial qubit placement is four times worse than the best-case runtime given a good structure and initial qubit placement.

Figure 4.17 shows the best cell structure found after conducting a search of all $2 \times 2$, $2 \times 3$, $3 \times 2$ and $3 \times 3$ sized cells for the three different circuits. As can be seen in the figure, the best grid-based layout is dependent on what circuit will be run upon it. By varying the location of gates and communication channels, the tools found a layout tailored to the circuit requirements.

To evaluate the performance of our optimized datapaths, we compare our datapaths to the QPOS grid shown in Figure 4.14(b)[43] (the most recent published datapath proposal). We can construct the QPOS grid structure with our macroblocks using a $2 \times 2$ sized primitive cell as shown in Figure 4.18. The dark gray box highlights the primitive cell.

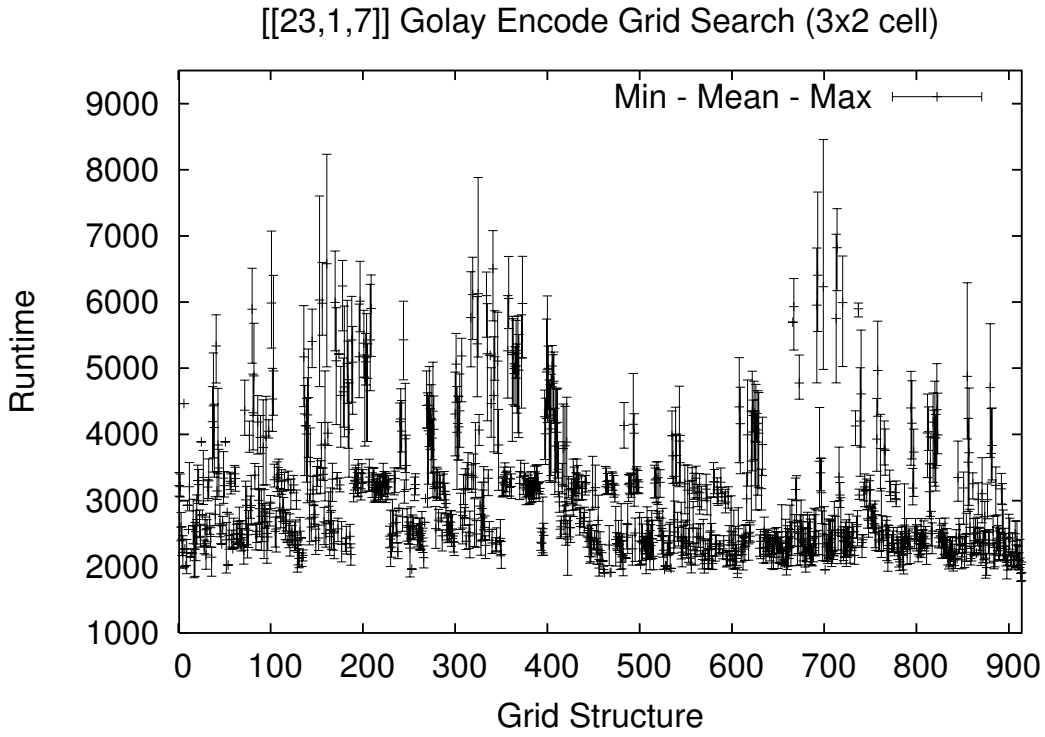



Figure 4.16: Variations in runtime of various grid-based physical layouts for $[[23, 1, 7]]$ Golay encode circuit. The datapath is constructed of $3 \times 2$ sized cells tiled to form a total area of 143 macroblocks. The X axis represents the different grid structures tested, only a small subset of the total is shown to demonstrate variability. For each grid structure the minimum, mean, and maximum time are plotted.

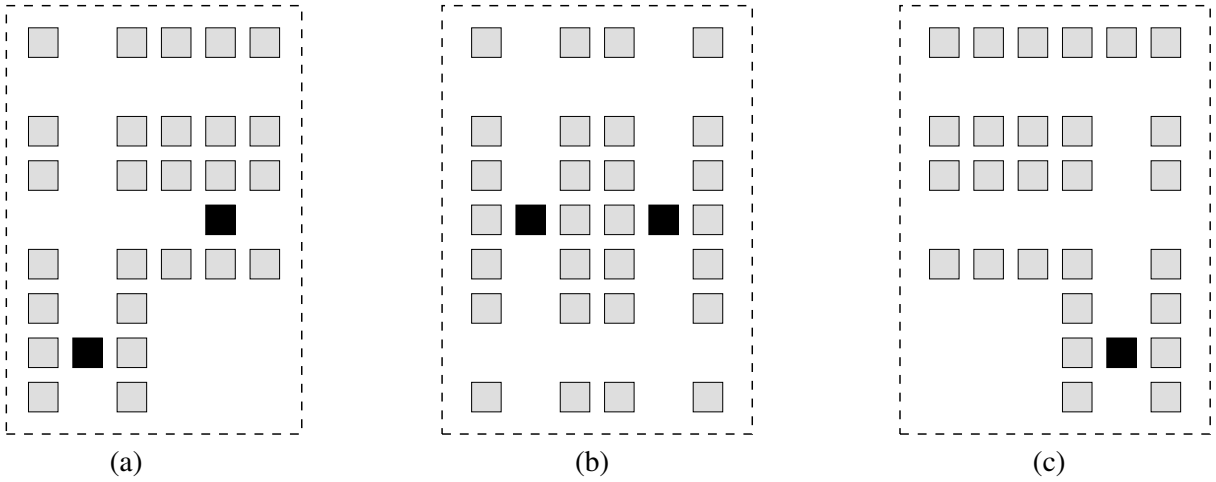



Figure 4.17: Comparison of the best $3 \times 2$ cell for two different circuits. (a) The best cell for the $[[23, 1, 7]]$ Golay encode circuit. (b) The best cell for the $[[7, 1, 3]]$ L1 correct circuit. (c) The best cell for the $[[7, 1, 3]$ L2 encode circuit.

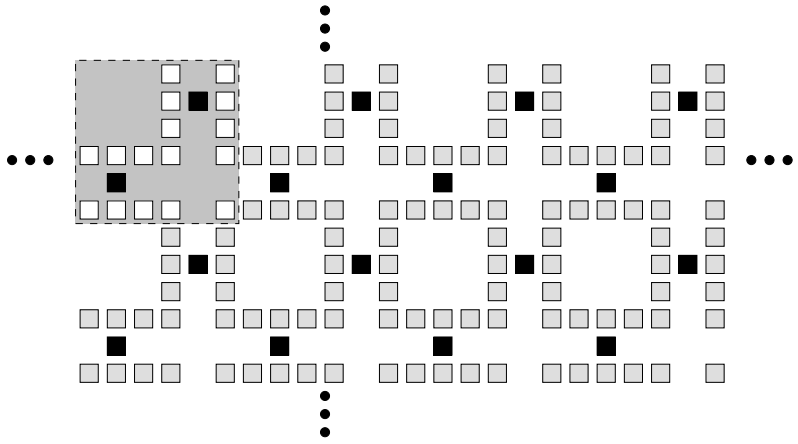


Figure 4.18: QPOS grid structure constructed by tiling the highlighted $2 \times 2$ macroblock cell. The cells can extend in all directions to create an arbitrarily sized layout.

| Circuit | Area | Datapath | Latency ($\mu s$) |
|---|---|---|---|
| $[[7, 1, 3]]$ L1 encode | 49 | QPOS Grid | 548.0 |
| | | Optimal Grid | 509.0 |
| $[[7, 1, 3]]$ L2 encode | 1365 | QPOS Grid | 2411.0 |
| | | Optimal Grid | 1367.0 |
| $[[23, 1, 7]]$ Golay encode | 575 | QPOS Grid | 2268.0 |
| | | Optimal Grid | 1801.0 |

Table 4.2: Latency results for the three error correction circuits we tested. In each case an exhaustive search for an optimal grid structure yielded a datapath with lower latency. The best grid structure found is shown in Figure 4.17.

In this case it is constructed of two `StraightChannel` blocks with gate locations (one rotated 90°) a `FourWayIntersection` block, and one empty block. The figure only shows a small portion of a layout constructed from 8 of these primitive cells, but any number of them can be tiled to form larger layouts.

In each case, our exhaustive search resulted in a cell structure that performed better than the QPOS grid. Table 4.2 shows the results of running the error correction circuits on the QPOS grid datapath and on the grid structure produced by our exhaustive search.

While this type of exhaustive search of physical layouts is capable of finding an optimal layout for a quantum circuit, it suffers from a number of drawbacks. Namely, as the size of the cell increases, the number of possible cell configurations grows exponentially. Searching for a good layout for anything but the smallest cell sizes is not a realistic option.

## 4.5 Qalypso Compute Regions

Since we are most interested in creating datapaths based on our Qalypso architecture, we use our tools to construct detailed layouts and schedules of operations within the

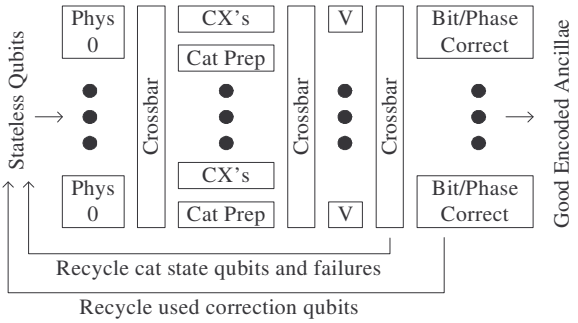



Figure 4.19: A fully pipelined encoded zero ancilla creation unit

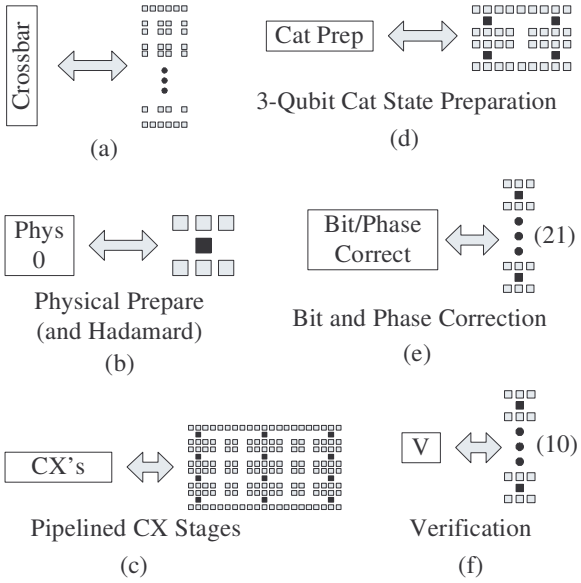



Figure 4.20: A layout of each unit in Figure 4.19.

major Qalypso components. A Qalypso datapath is constructed out of a set of compute regions connected via a communication network. The compute regions have two major components: a data area used for gate operations, and an QEC ancilla generation area used to create encoded ancilla. The first step to building a Qalypso datapath is to evaluate designs for these two major components.

We start by describing the structure of the ancilla generator units in Section 4.5.1 and follow that by discussing datapaths for the data portion of the compute region in Section 4.5.2.

### 4.5.1 Ancilla Generator Datapaths

In Section 4.4.3 we studied datapaths that perform in-place ancilla generation. In these structures the ancilla is generated in the module and moved out after generation is complete. After the encoded ancilla leaves, a new ancilla generation process can run. Rather than rely on the serial ancilla generation procedure, we instead use pipelined ancilla factories in our compute regions [34] as shown in Figure 4.19. Detailed layouts of the components are presented in Figure 4.20.

This pipelined ancilla factory has a number of benefits over the in-place version. The most prominent one being increased bandwidth. The process of pipelining allows us to create multiple ancilla simultaneously and output them at a higher rate than possible with a single in-place generator. It is possible to stack a number of in-place generators together to achieve the same bandwidth as a pipelined version. However, multiple in-place generators require us to move encoded ancilla much further to get to the final data. In contrast, the pipelined factory has a single output port that can be placed directly adjacent the data region. Thus, ancilla only need move a short distance from the generator to the

data.

Since ancilla generation is such a vital component of the circuit run, we manually design and layout the pipelined ancilla factories. We use our low-level scheduling tools to evaluate communication within the layouts and calculate the total bandwidth for the ancilla factories. These numbers are then fed into the CAD flow to evaluate the target quantum circuit.

### 4.5.2 Modeling Congestion

The data areas of the compute regions are where all the data qubits reside and where all the circuit operations take place. Communication operations within these regions play a large part is determining how the overall quantum circuit will perform. The flexibility of our CAD flow allows us to create arbitrary ion-trap datapaths and use them to build larger more complex circuits. To study the effects of congestion and movement latency, we schedule all the qubit movement and communication that occurs and determine performance.

In order to correctly determine the performance of these layouts we would schedule all the qubit movement and communication that occurs to study the effects of congestion and movement latency.

Unfortunately, as circuits become larger, the benefits of scheduling operations at a physical qubit level is quickly eclipsed by the time necessary to calculate the schedule. When scheduling requires multiple days to complete, the process of evaluating datapaths becomes extremely difficult. Since the main goal of our CAD flow is to allow the thorough study of many different parameters, we decided to sacrifice some low-level detail in an effort to decrease overall runtime of our analysis tools.

Rather than sacrifice all low-level detail to speed up circuit evaluation, we parametrize layout modules to create a communication model that we use to estimate overall circuit performance. This method allows us to quickly evaluate circuit while retaining the core effects resulting from communication needs. Since most of our circuits are built from logical qubits encoded as a set of physical qubits, we start by creating models of components at the logical qubit level.

For example, take the datapaths shown in Figure 4.21 and Figure 4.22. These are two potential datapaths for a compute region where logical qubits are encoded as three physical qubits. The datapath in Figure 4.21 is an area-efficient datapath. Logical qubits are grouped in a square with only a single communication channel separating the logical qubit regions. The alternate datapath shown in Figure 4.22 provides more communication channels and lays out the physical qubits in a line. The goal of this layout is to minimize communication delay to move a logical qubit. In datapath A, the physical qubits must serially move out of the logical qubit regions in order to reach another logical qubit. In datapath B, the qubits can all move simultaneously and remain in the line configuration.

Using these types of datapath inputs, our tools create a communication and congestion model of the compute region based on size and number of communications occurring at a given time. We start by constructing layouts with varying numbers of logical qubits.

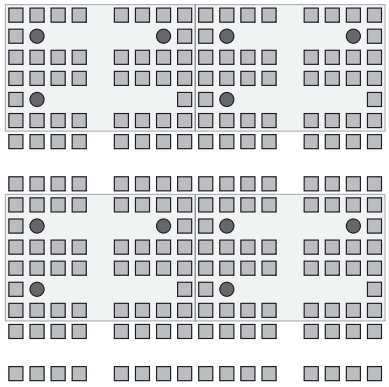


Figure 4.21: Compute Region Datapath A. In this example the logical qubit is encoded via three physical qubits and are laid out in an area-efficient manner.

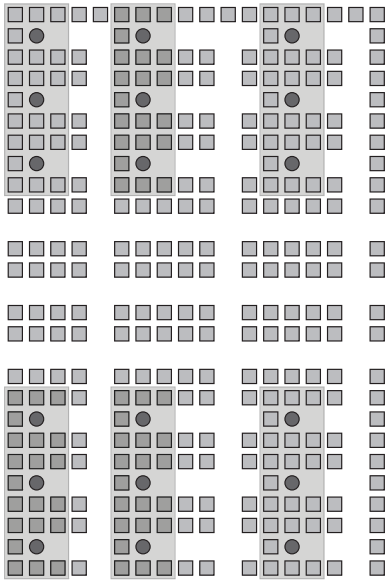


Figure 4.22: Compute Region Datapath B. In this example the logical qubit is encoded via three physical qubits and are laid out to optimize logical qubit movement.

With these layouts, we generate random communication operations and record the average amount of time qubits require to move within the datapath. As more communication operations occur at a given time, additional delay from congestion causes move operations to take longer and longer. We record the performance of the layout and use it when scheduling the higher-level operations (mapping and network routing).

Figures 4.23 and 4.24 illustrate the models we generate. In Figure 4.23 we see the performance of a compute region built using datapath A from Figure 4.21. The two lines on the graph correspond to a compute region sized to hold 16 logical qubits and a compute region sized to hold 36 logical qubits. Because the 16 qubit datapath is smaller, it starts with a lower average move latency when there is no congestion (0 other active moves). But, as illustrated in the graph, as we add more simultaneous move operations, the amount of congestion in the region causes the average move latency in the 16 bit datapath to increase much faster than in the 36 bit datapath.

Figure 4.24 contains the same setup except in this case for datapath B. Here we can clearly see the advantages to the additional communication channels in the datapath. When compared to datapath A, datapath B has a lower initial move latency (for the similar sized datapaths in bits) and additionally as more moves are performed the slope on the average move latency line remains much lower than that of datapath A due to the availability of more communication channels, limiting the amount of congestion-induced delays.

Our tools parametrize all the datapaths we study in this fashion and use the results

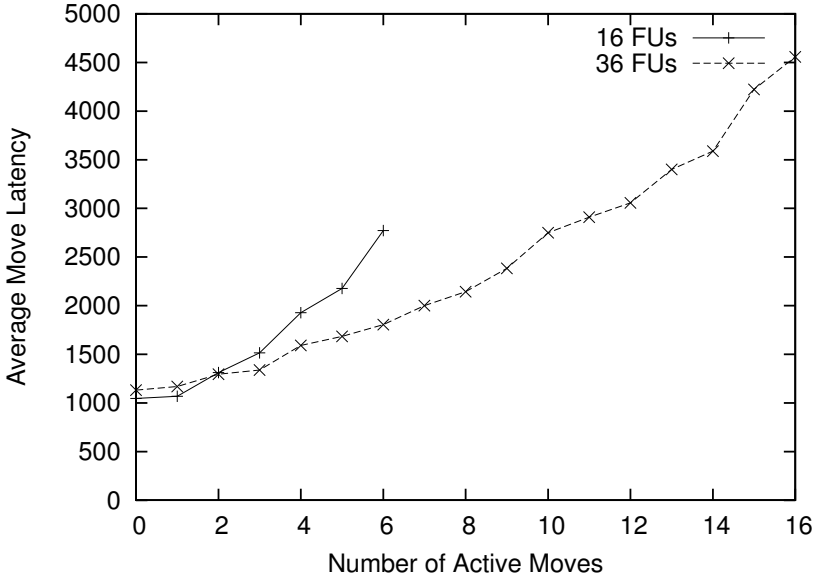



Figure 4.23: Average time (including congestion delays) to move a logical qubit given the number of active moves in the Compute Region for datapath A (Figure 4.21).

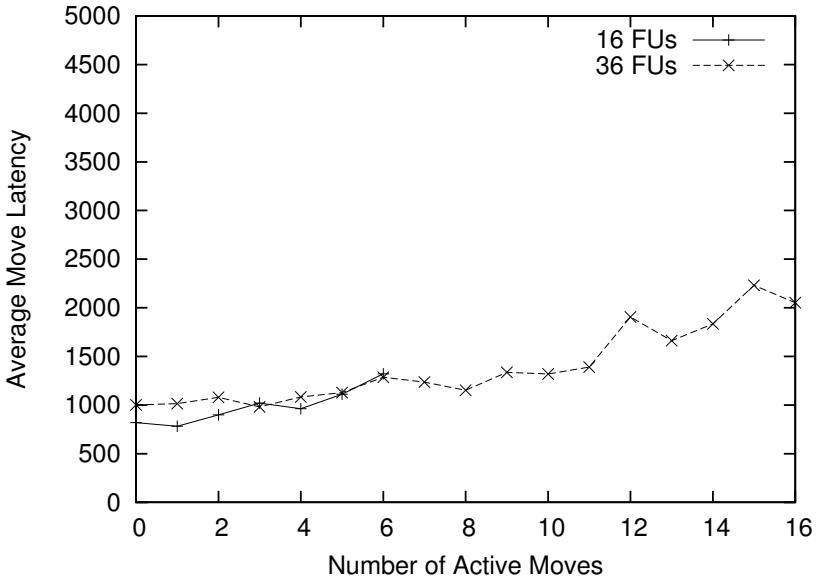



Figure 4.24: Average time (including congestion delays) to complete a logical qubit move given the number of active moves in the Compute Region for datapath B (Figure 4.22).

during the high-level mapping and routing phase. The results are cached and only updated when new datapaths are used or when lower-level parameters are changed. This approach allows us to properly evaluate the effects of communication and congestion in the low-level datapath without requiring us to schedule physical qubit movement when studying large circuits.

## 4.6 Summary

Building a low-level datapath out of ion-traps requires careful design of the layout and the control system. By using our macroblock abstraction, we are able to construct arbitrary datapaths where the control system is automatically assembled. Our abstractions allow us to study area and latency of ion-trap datapaths without relying on details of low-level implementation technology.

Using our layout and scheduling tools, we studied a number of different grid-based datapaths. We show that the structure used can dramatically affect overall latency of the circuit. Thus, it is important to tailor the datapath structure to the target quantum circuit in order to achieve the best performance. Our CAD flow uses these tools to evaluate different datapath structures and is designed to select the best datapath for the circuit under study.

# Chapter 5

# Optimizing Long-distance Quantum Communication

Being able to reliably and efficiently perform long-distance communication is vital to building a large-scale quantum computer. The quantum architecture we present in this work is constructed out of a set of compute regions connected by a quantum communication network, as shown in Figure 5.1. In this architecture all the computation is done on the qubits within the compute regions and all movement within a compute region is done using ballistic movement. Communication between compute regions makes use of the communication network and is performed via teleportation.

The basic process of teleportation within the communication network is shown in the right side of Figure 5.1. A data qubit $D$ enters the communication network and moves into a teleporter unit. The communication network is responsible for generating the EPR pair $E1, E2$ and distributing these qubits to the end points of the desired communication. When all qubits are in place, the teleport operation occurs, indirectly moving the state of $D$ into $E2$. After some classical information is transmitted to complete the teleportation, $D$ is now ready to move into the destination compute region.

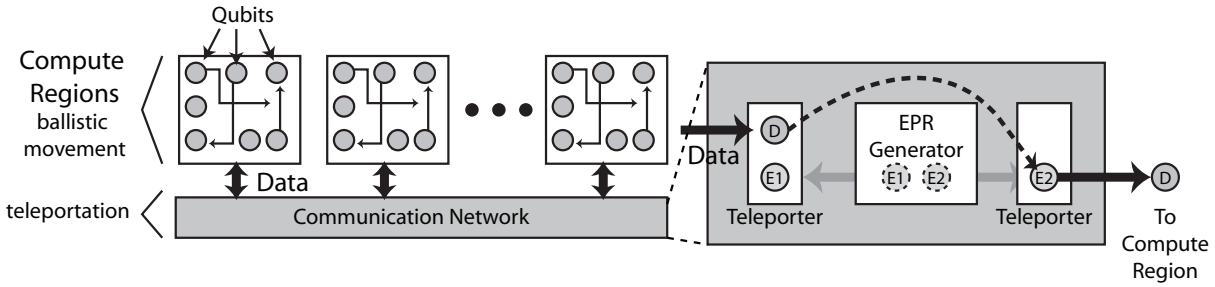



Figure 5.1: Overview of the Quantum Datapath. A set of Compute Regions are connected via a communication network. Within compute regions data uses ballistic movement. To communication between compute regions, data uses teleportation through the communication network. The right side of the figure shows a summary of the data teleportation process.

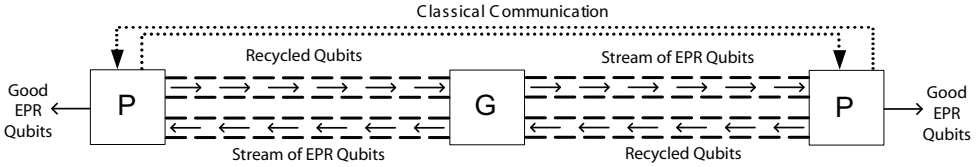



Figure 5.2: Ballistic Movement Distribution Methodology: EPR pairs are generated in the middle and ballistically moved using electrodes. After purification, high-fidelity EPR qubits are moved to the logical qubits, used, and then recycled into new EPR pairs.

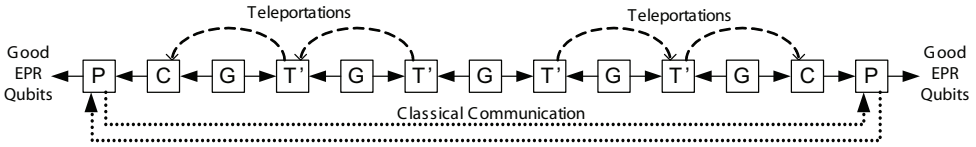



Figure 5.3: Chained Teleportation Distribution Methodology: EPR qubits generated at the midpoint generator are successively teleported until they reach the endpoint teleporter nodes before being ballistically moved to corrector nodes and then purifier nodes.

We favor teleportation for long-distance communication because it limits errors on the data qubits by drastically reducing the distance that they travel. Instead, the EPR qubits perform the long-distance movement in the data's place. Although naively it would appear that teleportation exposes the data to error by interacting data qubits with EPR qubits (which have accumulated error by long-distance communication), we utilize a process of *purification* [7] to remove noise from the EPR pair before performing teleportation. Specifically, purification improves the quality of some EPR qubits at the expense of others, allowing us to create relatively error-free EPR qubits for use in the data teleport.

## 5.1 Communication Network

The communication network's main task is to generate and distribute EPR pairs to the source and destination of requested communications. One option for EPR pair distribution is to generate EPR pairs at generator (G) nodes in the middle of the path and ballistically transport them to purifier (P) nodes that are close to the endpoints, as shown in Figure 5.2. Purification combines two EPR pairs to produce a single one of higher fidelity. For each qubit in the left purification (P) node, its entangled partner is in the right P node undergoing the same operations. For each purification performed, one classical bit is sent from each end to the opposite one. Discarded qubits are returned to the generator for reuse.

Another option is to generate an EPR pair and perform a sequence of teleportation operations to transmit these pairs to their destination. Correction information from a teleportation (two classical bits) can be accumulated over multiple teleportations and

performed in aggregate at each end of the chain. This process is depicted in Figure 5.3. A T' node contains units that perform the local operations to entangle qubits (step 2 in Figure 2.5), but no correction capability (step 4 in Figure 2.5). Instead, each T' node updates correction information and passes it to the next hop in the chain.

The path consists of alternating G nodes and T' nodes, with a C node and a P node at each end. Each G node sends EPR pairs to adjacent T' nodes. The EPR pairs generated at the central G node are moved ballistically to the nearest T' nodes, then successively teleported from T' node to T' node using the EPR pairs generated by the other G nodes. Since the EPR pairs along the length of the path can be pre-distributed, this method can improve the latency of the distribution if the T' nodes are spaced far enough apart.

Between each pair of "adjacent" T' nodes (as defined by network topology) is a G node continually generating EPR pairs and sending one qubit of each pair to each adjacent T' node. Thus, each T' node is constantly linked with each adjacent T' node by these incoming streams of entangled EPR qubits. Each G node is essentially creating a *virtual wire* which connects its endpoint T' nodes, allowing teleportation between them. By performing purification at either end of a virtual wire, we can increase the fidelity of the EPR pairs, effectively amplifying the connection between endpoints; we will investigate this effect in Section 5.2.3.

To permit general computation, any functional unit must have a nearby T' node (although they may be shared). This structure implies the necessity of a connected grid of T' nodes across the chip, which are linked by virtual wires. The exact topology is an implementation choice; one need not link physically close or even nearby T' nodes, as long as enough channels are included to allow each G node to be continuously linked to the endpoint T' nodes of its virtual wire with a steady stream of EPR qubits. Thus, any routing network could be implemented on this base grid of T' nodes, such as a butterfly network or a mesh grid.

### 5.1.1 Structuring Global Communication

As we discussed in Section 4.2, the process of moving quantum bits ballistically from point to point presents a challenging control problem. Designing control logic to move ions along a well-defined path appears tractable. However, controlling every electrode to select one of many possible paths becomes much more complex. Thus, we can benefit from restricting the paths that ions can take within our quantum computer. Such a tractable control structure will involve a sequence of "single-path" channels (much like wires in a classical architecture) connecting router-like control points.

Internally, we structure the communication network similar to that of a classical interconnection network using a mesh grid [1, 18] of routers connected to the various compute regions. In our case, the routers contain the teleporter and purifier units and the links consist of the EPR generators and channels to move qubits. We choose a mesh grid network topology (as opposed to higher dimension topologies) due to the planar nature of ion-trap datapaths. Current technology implementations do not have the luxury of stacking ion trap channels in multiple layers. Therefore, any link connecting non-neighboring

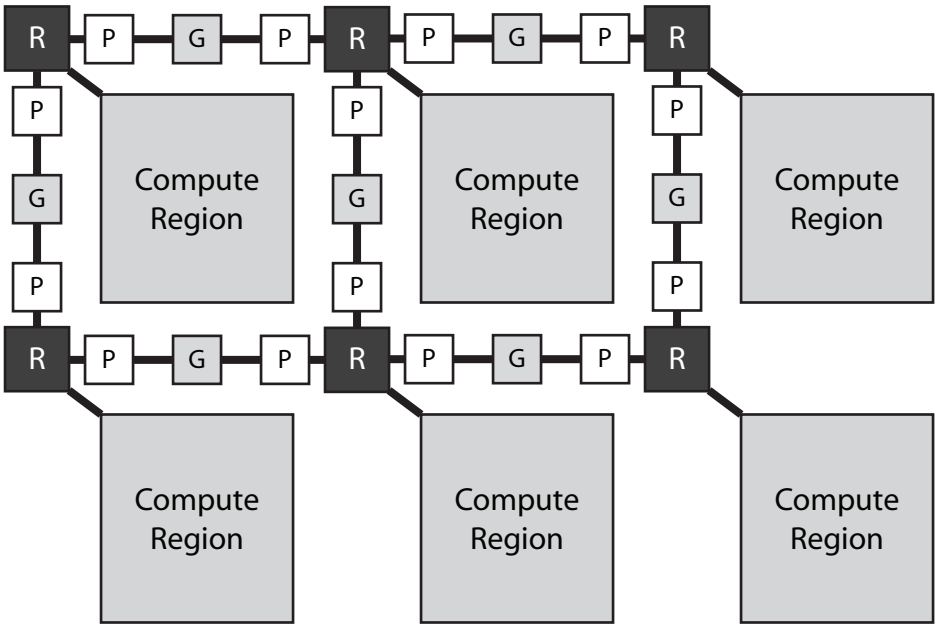



Figure 5.4: Communication network structure. The network is constructed as a mesh grid of routers connected via purifier-generator-purifier links. Each router has a local compute region where all the data operations occur.

routers would have to cross through existing links thereby increasing congestion and complicating control. Further, the distance traveled in a two-dimensional mesh is no worse than a factor of $\sqrt{2}$ over the shortest possible distance.

Figure 5.4 illustrates the general network structure. Each compute region connects to the network at a router. Routers contain teleporters that are used for both EPR teleport chaining and final data qubit teleportation. Additionally, routers have purifier units to purify the final EPR bits that will eventually be used for data teleportation. The neighboring routers are connected via links composed of centrally located generators connected to link purifiers (to purify EPR bits used within the link) as shown in Figure 5.2.

### 5.1.2 Terminology

The process of moving a logical qubit from one compute region to another via the network requires the coordination of a number of components within the network. To explain the details of communication with the network we must first present some terminology:

**Link:** A link connects two neighboring routers in the network. The link is composed of a set of generators and purifiers to link the two routers, along with the ballistic movement channels.The purifiers at either end of the link can be viewed as decreasing noise observed when using the link to teleport quantum information.

**Path:** A path is defined as a sequence of routers and links that connect them.

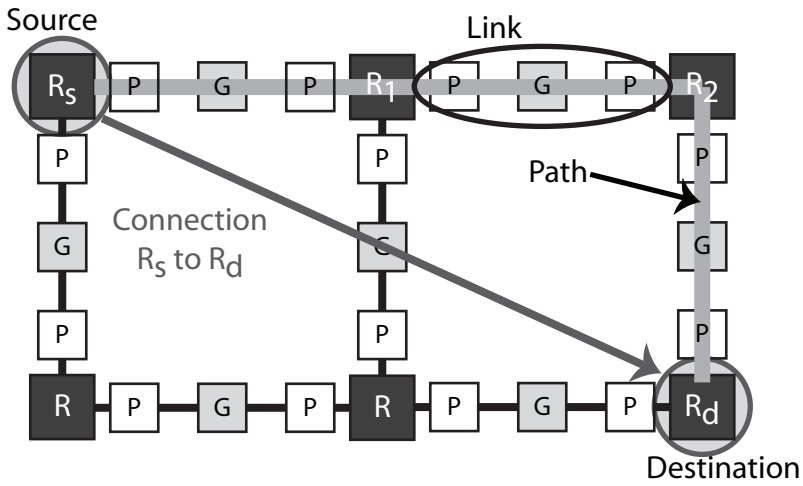



Figure 5.5: Illustration of network terminology. A network Connection from $R_s$ to $R_d$ is performed via the Path $R_s, R_1, R_2, R_d$. A Link is the P-G-P connection between routers.

**Connection:** A connection in our network links a source and destination router via some internal path. Two connections with the same source and destination router may use differing paths.

Figure 5.5 illustrates these terms. In it we see a connection from source router $R_s$ to destination router $R_d$. By itself, the connection only tells us the source and destination to which the final data EPR bits must eventually be routed. For this specific case, the connection is implemented as the path $R_s, L_1, R_1, L_2, R_2, L_3 R_d$. All of the purifier-generator-purifier connections between routers are links. Note that this connection can also be created along a number of other paths and our network provides flexibility along these lines.

### 5.1.3 Metrics

Within the network we will study various approaches to distributing EPR pairs keeping the following metrics for connection setup in mind:

**Fidelity:** Both ballistic transport and teleportation cause qubits to decohere. The architectural design must take into account the number of operations each qubit will undergo and the resulting chance for errors. As mentioned in Section 2.2.3, we will use fidelity to characterize the effects of errors on quantum information. Fidelity is a measure of the difference between the desired and actual state of a qubit.

**EPR Pair Count:** While most operations cause qubits to decohere, purification decreases error on one EPR pair by sacrificing one or more other pairs. The EPR Pair Count is a measure of the number of EPR Pairs that must be transmitted to either end of a connection in order to reach a desired level of EPR fidelity. The more error that is accumulated during connection setup, the more pairs that will need to be transported to the endpoints to achieve the desired EPR fidelity.

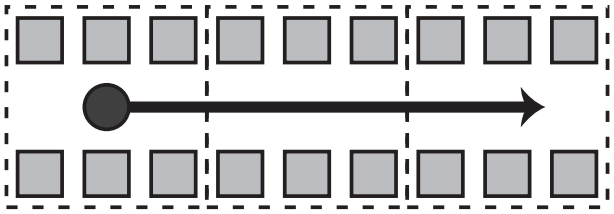


Figure 5.6: Ballistic Movement Model. Communication channels are constructed as a sequence of macroblocks. The fidelity of a qubit and time to move the qubit is directly proportional to the distance (in macroblocks) moved.

**Latency:** Logical communication set-up time determines how far in advance EPR distribution must occur. The latency of a connection is defined as the time to purify all the link level bits, chain teleport the data EPR bits, and perform the final data teleportation.

In addition, we keep the following more global architectural metrics in mind:

**Quantum Resource Needs:** The quantum datapath resource needs (quantity of each component) affect chip area and thus communication distance.

**Classical Control Complexity:** Generation, ballistic movement, teleportation and purification must each be controlled classically, so the classical control requirements vary with communication methodology.

**Runtime:** Ultimately, we want to know the impact of long-distance communication setup on execution time.

## 5.2 Network Communication Models

To properly evaluate the architecture of the communication network we start by introducing communication models for ballistic movement, teleportation and purification. Our models are centered around studying the *fidelity* (see Section 2.2.3) of the qubits involved in the communication operations.

### 5.2.1 Ballistic Transport Model

Ballistic movement is the low-level method for moving qubits within ion trap datapaths. In ballistic movement, the fidelity of a bit going through the ballistic channel and the latency of the movement operation is proportional to the distance moved. Figure 5.6 shows an example of a small communication channel constructed out of three macroblocks ($D = 3$).

The fidelity of the qubit after a move of distance $D$ is described by:

$$F_{new} = F_{old}(1 - p_{mv})^D \tag{5.1}$$

where $F_{old}$ is the fidelity of the bit at the beginning of the move and $p_{mv}$ is the probability of error for a single macroblock move. The time to perform ballistic movement is given in time per macroblock moved through and from Table 2.1 is $0.2\mu s/cell$.

$$t_{ballistic} = t_{mv} \times D \tag{5.2}$$

### 5.2.2 Teleportation Transport Model

Unlike the model of ballistic movement the fidelity of a qubit teleportation is more complicated because it involves a combination of single and double qubit gates ($p_{1q}, p_{2q}$) and qubit measurement ($p_{ms}$) [22]:

$$\begin{aligned} F_{new} &= \frac{1}{4}\left(1 + 3(1-p_{1q})(1-p_{2q})\frac{(4(1-p_{ms})^2 - 1)}{3}\right. \\ &\times \left.\frac{(4F_{old}-1)(4F_{EPR}-1)}{9}\right) \end{aligned} \tag{5.3}$$

Where $F_{new}$ is the data fidelity after the teleportation, $F_{old}$ is the data fidelity before teleportation and $F_{EPR}$ is the fidelity of the EPR bits used. The fidelity after a teleportation is dependent on the EPR pair fidelity and the data fidelity before teleportation.

Although ballistic movement error does not appear directly in this formula, it should be mentioned that the fidelity of the EPR pair will be degraded while being distributed to the endpoints of the teleportation channel. Thus, even though the qubit undergoing teleportation incurs no error from direct ballistic movement, there is still fidelity degradation due to EPR pair distribution. Thus, it is important to produce the highest fidelity EPR pairs that we can.

We produce EPR pairs from two qubits initialized to the zero state using a few single and double qubit gates. The fidelity of an EPR pair immediately after generation is:

$$F_{gen} \propto (1-p_{1q})(1-p_{2q})F_{zero} \tag{5.4}$$

$F_{zero}$ is the fidelity of the starting zeroed qubits. Generation time involves one single and one double qubit gate. As mentioned in Table 2.1, this time is projected to be $21\mu s$.

If we assume that EPR pairs are already located at the endpoints of our channel, teleportation time is given in Table 2.1 as $122\mu s$ and has the form:

$$t_{teleport} = 2t_{1q} + t_{2q} + t_{ms} + t_{classical\ bit\ mv} \times D \tag{5.5}$$

Because the time scale of quantum operations is on the order of $\mu s$, we assume the amount of time required to transmit classical data ($t_{classical\ bit}$) does not add appreciably to the teleportation and purification process.

### 5.2.3 EPR Purification Model

As shown by Equation 5.3, the fidelity of the EPR pairs utilized in teleportation ($F_{EPR}$) has a direct impact on the fidelity of information transmitted through the teleportation

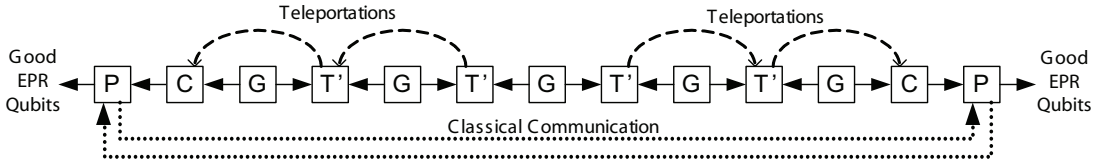



Figure 5.7: Chained Teleportation Distribution Methodology: EPR qubits generated at the midpoint generator are successively teleported until they reach the endpoint teleporter nodes before being ballistically moved to corrector nodes and then purifier nodes.

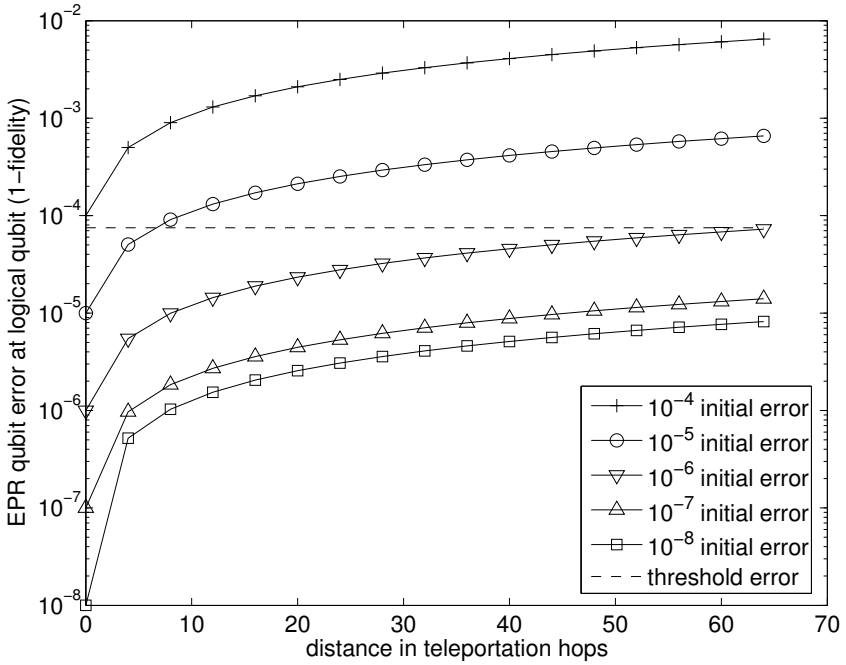



Figure 5.8: Final EPR error (1-fidelity) as a function of number of teleportations performed, for various initial EPR fidelities. The horizontal line represents the minimum fidelity the EPR pair must be at to be suitable for teleportation of data qubits, $1-7.5*10^{-5}$

channel. Since EPR pairs accrue errors during ballistic movement, teleportation by itself is not an improvement over direct ballistic movement of data qubits unless some method is used to improve the fidelity of EPR pairs.

Purification combines two lower-fidelity EPR pairs with local operations at either endpoint to produce one pair of higher fidelity; the remaining pair is discarded after being measured. Figure 5.9 illustrates this process, which must be synchronized between the two endpoints since classical information is exchanged between them. On occasion, both qubits will be discarded (with low probability).

The purification process can be repeated in a tree structure to obtain higher fidelity EPR pairs. Each *round* of purification corresponds to a level of the tree in which all EPR pairs have the same fidelity. Since one round consumes slightly more than half

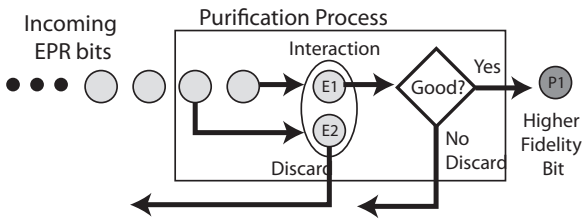



Figure 5.9: Purification Process. Two EPR bits undergo local operations and exchange classical information with the partner unit. One bit is immediately discarded. The remaining bit is retained if good, otherwise it is also discarded.

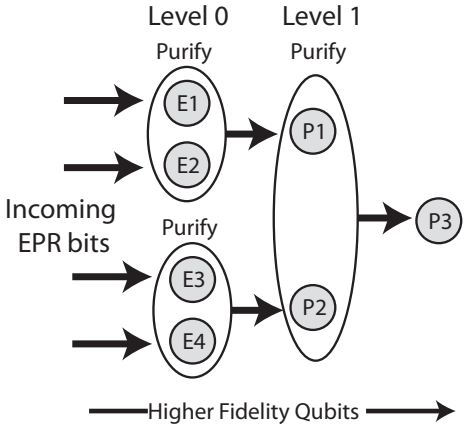



Figure 5.10: Tree Purification Process. Level 0 qubits are purified to form Level 1 qubits which are then purified to create a Level 2 qubit.

of the remaining pairs, resource usage is exponential in the number of rounds. There are two similar tree purification protocols, the DEJMPS protocol [19] and the BBPSSW protocol [7]. The analysis of the DEJMPS protocol provides tighter bounds which assures faster, higher fidelity-producing operation compared to the BBPSSW protocol. The effects are significant, implying that purification mechanisms must be considered carefully[1].

Figure 5.11 shows error rate as a function of number of purification rounds. The BBPSSW protocol takes 5-10 times more rounds to converge to its maximum value as the DEJMPS protocol. Since EPR pair consumption is exponential in number of rounds, the purification protocol has a large impact on total EPR resources needed for communication. Other features of Figure 5.11 to note are that DEJMPS has higher maximum fidelity and converges to maximum fidelity faster than BBPSSW (possibly because BBPSSW partially randomizes its state after every round).

Finally, the time to purify a set of EPR qubits is dependent on the initial and desired fidelity. The time to complete one round of purification is $121\mu s$ from Table 2.1:

$$t_{purify\ round} = t_{2q} + t_{ms} + t_{classical\ bit} \tag{5.6}$$

where $t_{2q}$ is the time to perform a two qubit gate, $t_{ms}$ is the time to measure a qubit, and $t_{classical\ bit}$ is the time to transmit a classical bit to the purifier on the other end.

### 5.2.4 Communication Model Analysis

When studying error and fault tolerance in quantum computers, researchers are often interested in learning the threshold for error that still permits sustainable quantum com-

[1]Dur also proposes a linear approach to purification [22]; unfortunately, it appears to be sensitive to the error profile. We will not analyze it here.

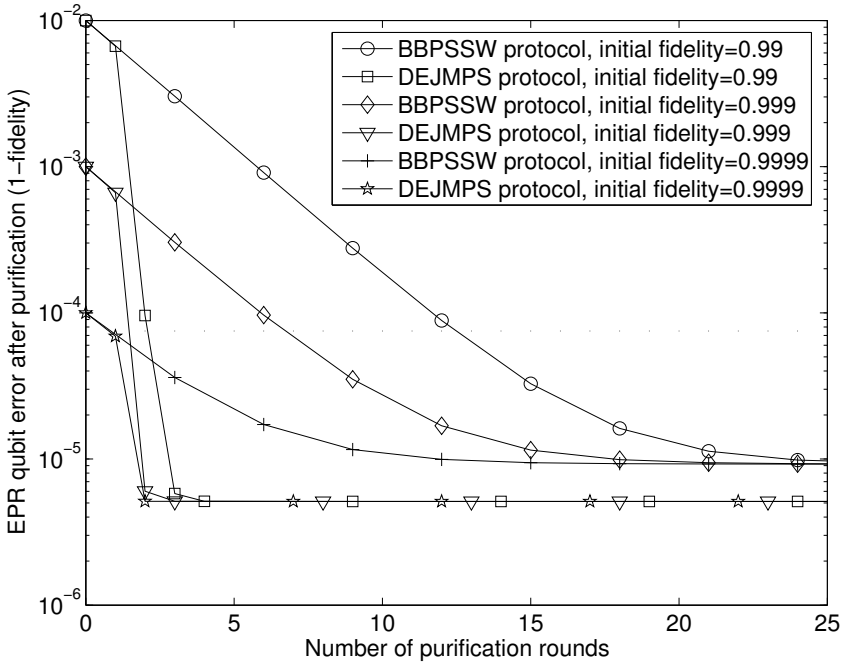



Figure 5.11: Error rate (1-fidelity) for surviving EPR pairs as a function of the number of purification rounds (tree levels) performed by the DEJMPS or BBPSSW protocol. Lower is better.

putation. Essentially, we want to know how much error an operation (gate or movement) is permitted to introduce into the system while still maintaining fault tolerance. The most recent version of the threshold theorem for fault-tolerant quantum computation indicates that data qubit fidelity must be maintained above $1 - 7.5 * 10^{-5}$ [65]. Any operation in the system that degrades fidelity past this value is unlikely to operate successfully as operations used to perform error correction will introduce more error into the system than they remove. Therefore, our goal is to construct a communication system that maintains qubit fidelity above this threshold.

The microarchitectures we study use teleportation based interconnect for long range qubit communication. In long range communication, the preservation of data qubit fidelity, is our highest priority. Therefore, we choose to transport all data by way of single teleports, since this approach introduces the minimum error from ballistic movement. Using teleportation necessitates the distribution of EPR pair qubits to communication endpoints. Since data qubits interact with these EPR pairs, the above threshold must be imposed on them to avoid tainting the data.

Two options present themselves for distributing high-quality EPR pairs to channel endpoints. First, one could ballistically move the EPR pairs to the endpoints, which is preferable to moving data ballistically because EPR pairs can be sacrificed if they accumulate too much error. Second, one could route EPR pairs through a series of teleporters, as shown in Figure 5.3. While preserving fidelity of our data states is top priority, when dealing with less precious EPR pairs, we do not have to adhere to strict

maximal fidelity preserving distribution methods since we can use purification to amplify the fidelity of EPR pairs once they reach their destination. In the rest of this section, we will investigate the trade-offs between ballistic distribution and chained teleportation distribution of EPR pairs.

**Fidelity Difference:** The final fidelity of these two techniques is approximately the same. Conceptually, the final EPR pair either directly accumulates movement error (through ballistic movement) or is interacted with several other EPR pairs to teleport it to the endpoints and these intermediate EPR pairs have accumulated the same distance ballistically. By interacting with intermediate pairs, the final pair accumulates all this error. This statement assumes that the fidelity loss from gate error is much less than the loss due to ballistic movement, which is the case for ion traps, as shown in Table 2.1 (for two teleporters spaced 100 macroblocks apart, ballistic movement error equals $1 - (1 - 10^{-6})^{100} \approx 10^{-4}$ compared to $10^{-7}$ for a two-qubit gate error).

Long-distance distribution of EPR pairs can severely reduce the fidelity of the EPR pairs arriving at a functional unit for data teleportation, as shown in Figure 5.8. In order to process 1024 compute regions, we could imagine arranging them on a square 32x32 grid, in which the longest possible Manhattan distance is 64 compute region lengths. If we assume that we have teleporter units at every compute region, EPR pair distribution could require up to 64 teleports. From the figure, teleporting 64 times could increase EPR pair qubit error by a factor of 100. The dotted line represents the threshold at which the EPR pairs must be in order to not corrupt the data qubit when teleporting it. In order to preserve data fidelity, we must use EPR pair purification. One way to think about this process is to stitch Figures 5.11 and 5.8 side-by-side, so that EPR pairs accumulate error (degrade in fidelity) as they are teleported and then purified back to a higher fidelity at the endpoints before being used with data.

**Latency Difference:** Equation 5.2 shows a linear dependence on distance for ballistic movement latency. Equation 5.5 also shows that teleportation has a linear dependency on distance as well, but the constant in this case is for the necessary classical communication. We assume classical information can be transferred on a time scale orders of magnitude faster than the quantum operations.

If teleportation is considered performed in near constant time, then we would like to know the distance crossover point where teleportation becomes faster than the equivalent ballistic transport. From Table 2.1, teleportation takes about $122\mu s$ while ballistic movement takes $0.2\mu s$ per macroblock. Thus for a distance of about 600 macroblocks, teleportation is faster than ballistic movement. We assume our communications fabric to be a 2-D mesh of teleporter nodes and use 600 cells as the distance that each teleportation "hop" travels. Allowing teleportations of longer distances would further reduce communication latency in some cases but would then require more local ballistic movement to get an EPR pair from the nearest teleporter to its final destination.

### 5.2.5 Purification Resources

Earlier in this section, we noted that when we purify a set of EPR pairs, we measure and discard at least half of them for every iteration. This EPR overhead means that to perform $x$ rounds, we need more than $2^x$ EPR pairs to produce a single good pair.

To measure EPR resource usage, we count the total number of pairs used *over time* to move a level 2 [60] error corrected logical data qubit between endpoints. The teleportation requires us to transport 49 physical data qubits some distance by way of teleportation. We find that the total number of EPR qubits necessary to move a datum critically affects the data bandwidth that our network can support. This metric differs from that used in a number of proposals for quantum repeaters which focus on the layout of a quantum teleporter and are most concerned with spatial EPR resources, i.e. how much buffering is necessary for a particular teleporter in the network [14]. We will show that our design is fully pipelined, and therefore only a small number of qubits must be stored at any place in the network at any time.

We saw in Figure 5.11 that if we start at a relatively low fidelity and try to obtain a relatively high fidelity, we could need more than a million EPR pairs to produce a single high fidelity pair using the BBPSSW protocol. Therefore we use the DEJMPS protocol in all further analysis. Even though the DEJMPS protocol converges to good fidelity values much quicker, the exponential increase in resources for each additional round performed means we must be careful about how much error we accumulate when distributing EPR pairs. We will also show that the point in the datapath at which purification is performed can have a dramatic impact on total EPR pairs consumed. We consider three reasonable options:

**Endpoints only:** Purify only at the endpoints, immediately before using EPR pairs to teleport data.

**Virtual wire:** Purify EPR pairs which create the links between teleporters, namely the constant stream of pairs from a G node to adjacent T' nodes. The result is higher fidelity qubits used for chained teleportation.

**Between teleports:** Purify EPR pairs *after* every teleportation; this method purifies qubits that are being chain teleported rather than qubits assisting the chained teleportation.

We now model the error present in our entire communication path. Assuming the EPR pairs at the logical qubit endpoints must be of fidelity above threshold, we determine the number of EPR pairs needed to move through different parts of the network per logical qubit communication.

**Total EPR Resources:** Figure 5.12 shows that the Endpoints Only scheme uses the fewest total EPR resources. This conclusion is evident if we refer back to Figure 5.11, where purification efficiency asymptotes at high fidelity; thus, purifying EPR pairs of lower fidelity shows a larger percentage gain in fidelity than purifying EPR pairs of high fidelity. From this figure, we can see that to minimize total EPR pairs used in the whole system, it makes sense to correct all the fidelity degradation in one shot, just before use.

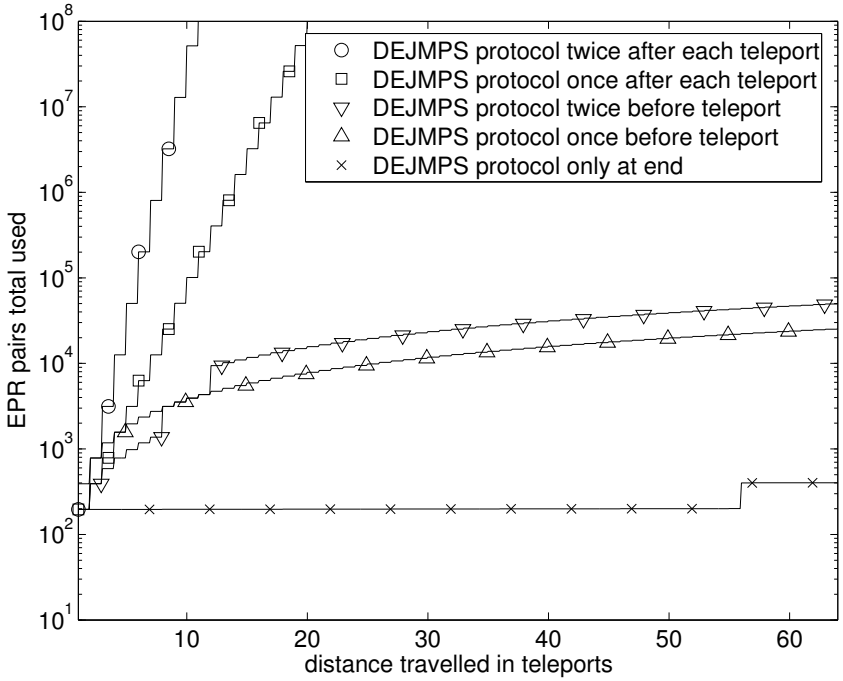



Figure 5.12: Total EPR pairs consumed as a function of distance and point at which purification scheme DEJMPS is performed.

**Non-local EPR Pairs:** Another metric of interest is to focus only on those EPR pairs that are transmitted to endpoints during channel setup (i.e. those that are teleported through the path). This resource usage is critical for several reasons: First, every EPR pair moved through the network consumes the slow and potentially scarce resource of teleporters; in contrast, the EPR pairs consumed in the process of producing virtual wires are purely local and thus less costly. Second, because of contention in the network, EPR pairs communicated over longer distances (multiple hops) place a greater strain on the network than those that are transmitted only one hop. The channel setup process can be considered to consume bandwidth on every virtual wire that it traverses. Third, the total EPR pairs transmitted to endpoints during channel setup consumes purification resources at the endpoints—a potentially slow, serial process.

Figure 5.13 shows that purifying EPR pairs after each teleport transmits many more EPR pairs than purifying at the endpoints (either with or without purifying the virtual wires). From this figure, we see that over-purifying bits leads to additional exponential resource requirements without providing improved final EPR fidelity[2]. Virtual wire purification improves the underlying channel fidelity for everything moving through the teleporters, thereby allowing less error to be introduced into qubits traveling through the channel. For a given target fidelity at the endpoints, virtual wire purification reduces the number of EPR pairs that need to move through the teleporters and also reduces the

[2]The authors of [14] claim that this nested purification technique (after every teleport) has small resource requirements; however, they count spacial resources rather than total resources over time.

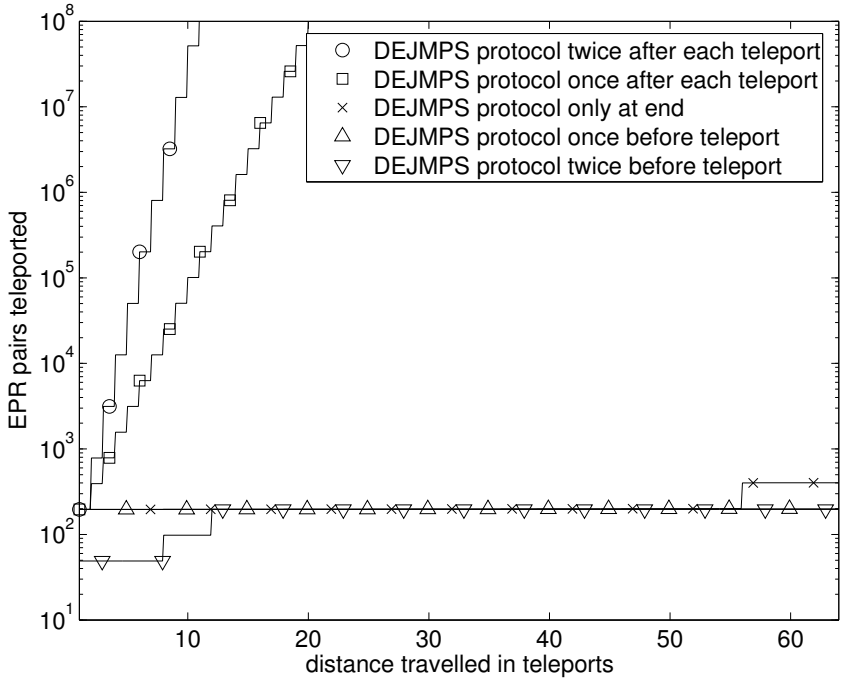



Figure 5.13: Total EPR pairs in teleportation channel as a function of distance and point in transport in which purification scheme DEJMPS is performed. The only 2 lines that change from Figure 5.12 are the purify before teleport cases.

strain on the endpoint purifiers.

To summarize, we have made the following design decisions based on fidelity and latency concerns:

**Teleport data always:** Data qubits sent to destination with single teleportation to minimize ballistic error.

**Teleport EPR pairs:** EPR pairs distributed to endpoints with teleportation, allowing pre-purification to increase the overall fidelity of the network.

**Purification before teleport and at endpoints:** Purify intermediate EPR pairs before they are used for teleportation as well as EPR pairs at the channel endpoints.

Finally, Figure 5.14 shows the sensitivity of the EPR resources necessary to sustain our previous error threshold goals as a function of the error of the individual operations like quantum gates, ballistic movement, and quantum measurement. The first thing to note are the abrupt ends of all the plots near $10^{-5}$. This value is the point at which our whole distribution network breaks down, and purification can no longer give us EPR pairs that are of suitably high fidelity (above $1 - 7.5 * 10^{-5}$). The fact that all the purification configurations stop working for the same error rate is due to the fact that the purification schemes we investigated are limited in maximum achievable fidelity by operation error rate and not the fidelity of incoming EPR pairs (unless the fidelity is *really* bad). Throughout

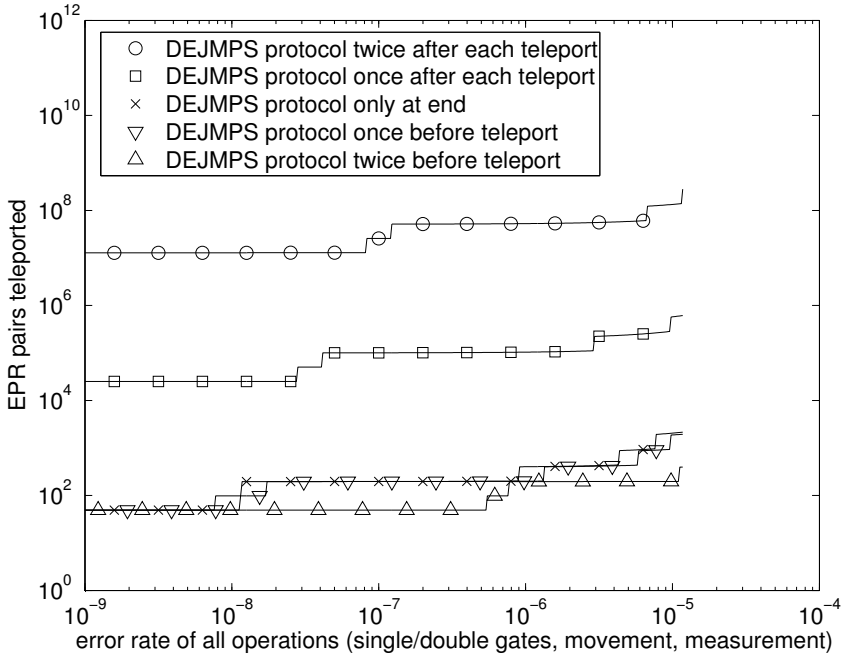



Figure 5.14: Number of EPR pairs that need to be teleported to support a data communication within the error threshold. All error rates are set to the rate specified on the x-axis.

the regime at which our system *does* work however, the total network resources only differ by a factor of up to 100 for a 10,000 times difference in operation error rate.

## 5.3 Network Connections and Control

In our network, a single connection between routers requires the coordination of a number of different components in the system. This coordination and control is conducted through a classical control network which runs parallel to the quantum datapath. A detailed sequence of operations to set up a communication connection within the network is show in Figure 5.15. Here we see all the classical control messages that must be transmitted between the various components in order to establish and perform the connection.

We start this section by describing the the control messages used to perform chain teleportation (the basis of how our EPR bits are distributed to the destination routers). We will then describe how the various stages of the network connection enable this process to occur.

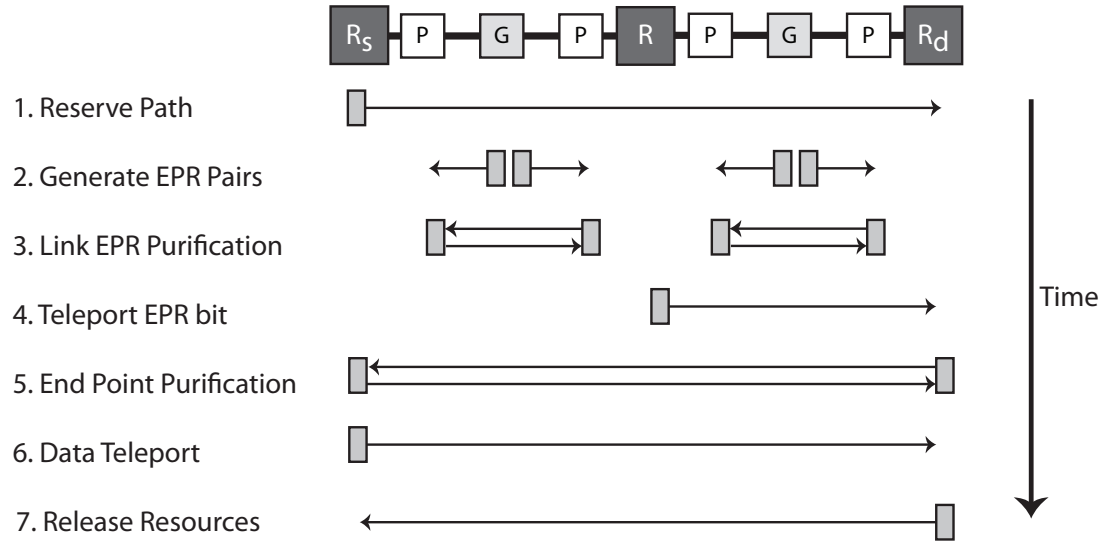



Figure 5.15: Classical control messages to setup and use a network connection. Each stage in the process requires the passing of classical control messages to synchronize the operations. The process is described in detail in Section 5.3.

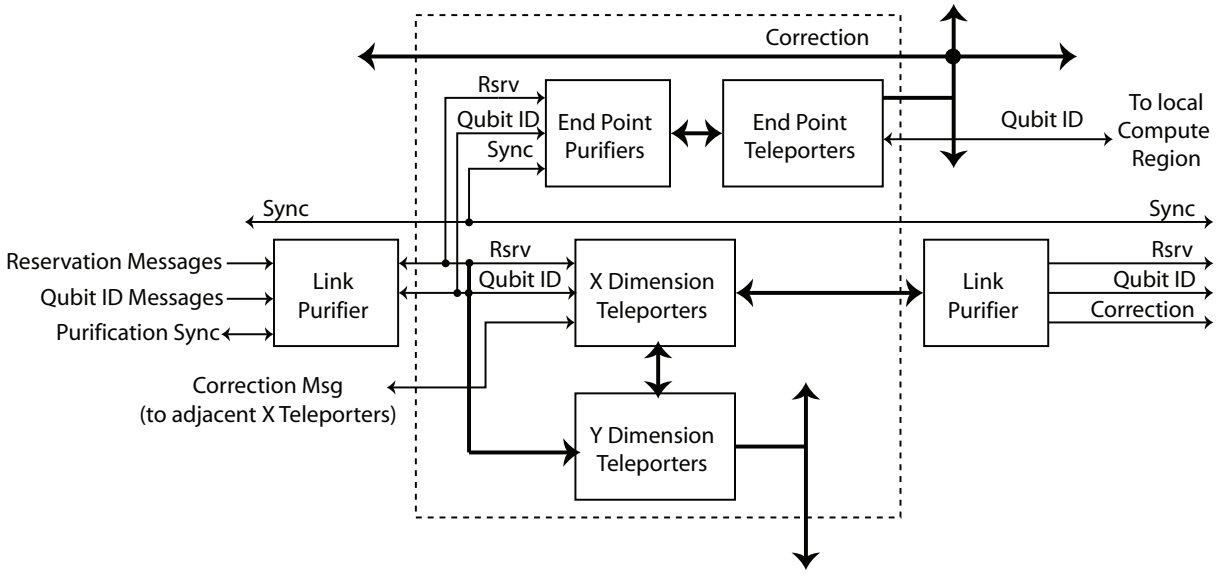



Figure 5.16: Control interface for Routers. Routers have control signals connected to the local compute region, link purifiers, and neighboring routers. The signals for the Y teleporters are identical to the X teleporters and are summarized as bold lines. The link purifier modules for the Y dimension are omitted for clarity.

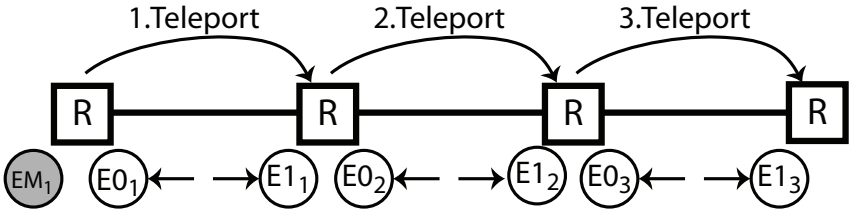



Figure 5.17: Chain Teleport Procedure. To get the master EPR bit from the source to the destination router a sequence of chained teleports are performed. $EM_1$ is first teleported using $E_1$, then $E_2$, and finally $E_3$.

### 5.3.1 Chain Teleportation

We use a process of chain teleportation (Figure 5.7) to transport the EPR bits from the master generator link to the final routers. Figure 5.17 shows one half of the teleportation operation where the goal is to move one half of the master EPR pair $EM_1$ to the destination. This task is accomplished using three teleport operations in sequence.

One way to perform the chained teleport operations is to perform the full teleport circuit (Figure 5.18(a)) in each router. At each router two classical bits would be sent from the previous router and used to perform the controlled-$X$ and controlled-$Z$ gate to correct the data. This method is illustrated in Figure 5.18(b). In Figure 5.18(c) we show an expansion of the teleport module. As can be seen in the figure, after two hops we must execute 4 correction gates, one controlled-$X$ and one controlled-$Z$ for each hop. If we have to perform 3 hops, we would require 6 correction operations, and so on. Rather than introduce correction operations for each hop in the path, we optimize the process in order to reduce the number of gates necessary at the end point to three: controlled-$X$, controlled-$Z$ and a controlled-$Sign$ gate.

To understand the optimization, we first start with the sequence of correction operations that occur assuming a two-hop connection:

$$X_{cx_2} Z_{cz_2} X_{cx_1} Z_{cz_1}$$

Where each controlled-$X$ and controlled-$Z$ gate is controlled by their respective classical control bit indicated in the subscript ($\mathtt{cx}_2, \mathtt{cz}_2, \mathtt{cx}_1, \mathtt{cz}_1$). Using the quantum gate identity $ZX = -XZ$ we can rearrange the formula to read:

$$(-1)^{cz_2 \wedge cx_1} X_{cx_2} X_{cx_1} Z_{cz_2} Z_{cz_1}$$

We've added a controlled-$Sign$ operation that is performed if both $\mathtt{cz}_2$ and $\mathtt{cx}_1$ are enabled. From this point we make use of the following gate identities: $XX = I$ and $ZZ = I$. With these identities, we can reduce the sequence of gates to:

$$(-1)^{cz_2 \wedge cx_1} X_{cx_2 \oplus cx_1} Z_{cz_2 \oplus cz_1}$$

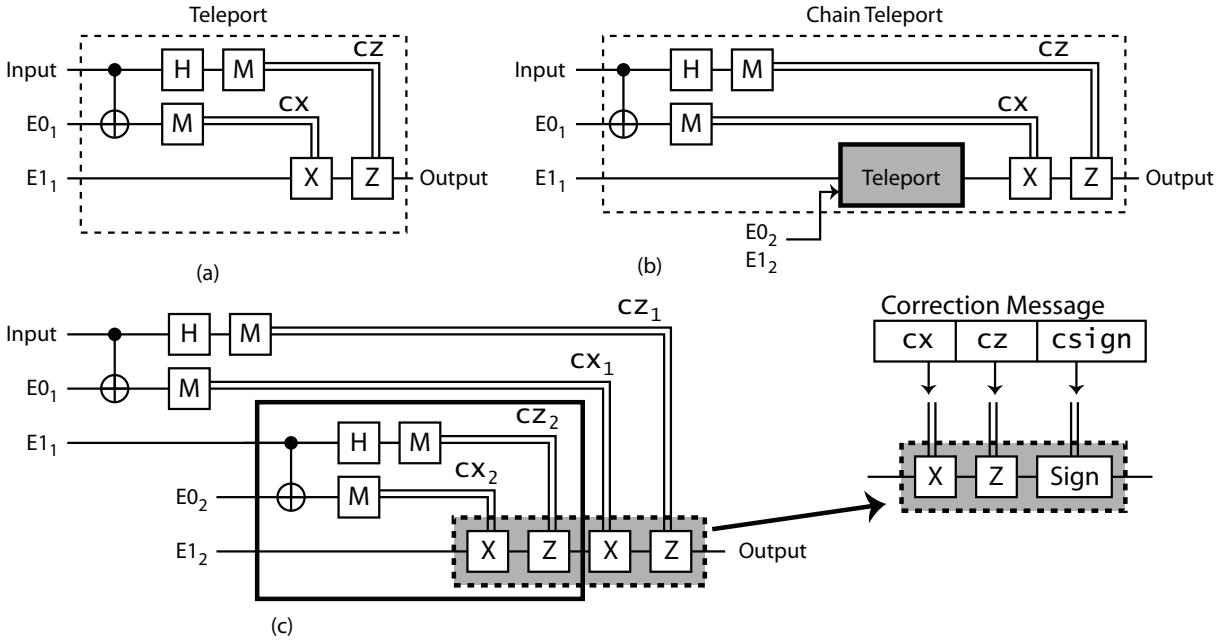



Figure 5.18: Chain Teleport Circuits. (a) The standard teleport circuit that uses $E0_1$ and $E1_1$ to teleport Input to Output. Classical bits `cx` and `cz` are transmitted to the destination and used to control the $X$ and $Z$ operations. (b) Circuit representation of chain teleportation where an additional teleport operation is inserted to teleport $E1_1$ to the destination (via $E0_2$ and $E1_2$). (c) Same as the circuit in (b) with the Teleport box expanded. This figure shows the controlled-$X/Z/X/Z$ correction operations that must occur at the destination to recreate the Input value. Our optimizations remove the multiple correction gates and instead collect the classical bits at each hop in the form of a correction message. At the destination, the correction message controls a single set of controlled-$X, Z, Sign$ gates.

Rather than performing the four controlled gates, we now only have to perform three controlled gates (Figure 5.18(c)). We can extend this process to support as many hops as necessary, and still only perform the three controlled gates at the destination. At each hop in the chain teleport process, we collect the correction information in a correction message. We start by creating a correction message at the source in the form `[cx, cz, csign]` where `cx`, `cz`, and `csign` start with the value 0. After each teleport hop the message is updated such that after hop $n$ the message becomes:

$$[\mathtt{cx}_{n-i} \oplus \mathtt{cx}_n,\ \mathtt{cz}_{n-1} \oplus \mathtt{cz}_n,\ \mathtt{sign}_{n-1} \oplus (\mathtt{cx}_{n-1} \wedge \mathtt{cz}_n)]$$

At the destination we use these correction bits to perform the final controlled-$X$, controlled-$Z$ and controlled-$Sign$ gates.

### 5.3.2 Path Reservation

A connection begins with a connection setup message issued by the high-level network control unit. The connection setup message contains a unique connection ID, the ID of the logical qubit to teleport, an ID of the master link whose EPR pair will be used for the final data teleport, and the full path through the network the connection will use (as determined by the routing phase described in Section 6.2. This setup message is initially issued to the source router. The source router reserves local resources (teleporters and end-point purifiers) and then passes the message on to the first link in the path.

When a link receives a connection setup message it reserves local resources for the connection (generator and link level purifiers) and passes the message on to the next router in the path. In this manner, the message is passed along to reserve resources until the destination router is reached.

Figure 5.16 shows how the reservation messages pass through the router interface. Reservation messages come through the link purifiers and enter the router to reserve EPR teleporters (X/Y dimension teleporters) and the end point resources (purifiers/teleporters). The reservation message is then passed on to the next router in the path.

The reservation message is passed along the connection path and initiates the link setup and EPR distribution described below.

### 5.3.3 Link Setup

Network links generate, distribute and purify EPR bits to connect adjacent routers and create the virtual wires. A link can operate in either direction, however at any given time a link will only operate in a single direction. The reservation message sets the link's direction according to the needs of the connection. Once the reservation message triggers the connection setup, the process begins. EPR pairs are created at the centrally located generator and moved to the link purifiers. Each EPR bit generated has an associated ID which allows the network to identify which EPR bits form entangled EPR pairs. As the EPR bits travel through the quantum datapath, this control message for the bit travels with it in the classical network (step 2 in Figure 5.15 ).

A qubit's message contains the ID assigned by the G node, the destination of this qubit, the destination of its partner (which is necessary for the purification steps at the endpoints), and space for the cumulative correction message described above. Each G and P node needs local classical control to determine how it handles qubits. G nodes have minimal intelligence. Each G node has one or more dedicated channels for sending fresh EPR pairs to its P node endpoints. When the P node blocks (due to being full), the channel blocks, and when the channel blocks, the G node stalls its generation. Thus, a G node is capable of determining whether it can fit more qubits on its outgoing channels.

When EPR bits reach the link level P nodes they are purified before entering the EPR teleporters. The process of purification takes two bits and outputs a single higher quality bit and discards the other, or with low probability discards both bits. Each end point of the link must synchronize to establish which bit is being purified and which bit is being discarded. The purifier uses the qubit message to notify the partner purifier about the

outcome of the purification process. This process is shown as step 3 in Figure 5.15. After purification, one or both of the bits are discarded and recycled back to the generator while the good (if purification was success) bit is moved into the adjacent routers.

### 5.3.4 EPR Teleportation

The EPR Teleportation stage takes the link-level EPR bits and teleports them via chain teleportation to the source and destination routers. The master link is specified by the high-level control and its identifier is contained in the initial path reservation message. This link will generate the EPR bits that will eventually be used for data teleportation. All the remaining links in the path will be creating EPR pairs to chain teleport the master EPR bits to the source and destination router.

The master link's generator creates EPR bits with the empty correction message as described in Section 5.3.1. After these bits are purified and they enter the EPR teleporters they are chain teleported to the end points of the path, one hop at a time. After each hop, the correction message is updated to reflect the new correction values that will be applied at the end points.

In each router, there are two banks of EPR teleporters, one for the X network dimension, and one for the Y network dimension (shown in Figure 5.16). Qubits are routed to the appropriate bank depending on what direction the connection is going. Each bank of teleports connect to the link purifiers to process the incoming EPR bits. Additionally, a control message interface connects the two banks with the banks in the neighboring routers to synchronize teleports and correction bits.

### 5.3.5 Data Teleportation

When the EPR bits reach the source and destination router for the connection they are moved into end-point purifiers. The EPR bits are purified according to the total distance travelled (in teleport hops). This purification process requires synchronization between the the end-point purifiers at the source and destination routers. Once the bits are sufficiently purified they are moved into the end-point teleporters within the router node. Only the source and destination router in the connection require the use of these end-point purifiers and teleporters. The EPR Teleportation process must remain active until the source and destination receive enough high-fidelity EPR bits to teleport the logical qubit (a number dependent on the desired error-correction encoding).

After the data teleport completes, the qubits are moved into the local compute region to perform their operations. Figure 5.16 shows the control interface for the end-point resources. End-point purifiers and teleporters are directly connected to the end-point purifiers and teleporters in the neighboring routers. This interface is used to synchronize purification and teleportation correction messages.

When the data exists the destination router, a path tear-down message is send backwards along the path to the source router. This message releases the resources along the path so following connections may use them.

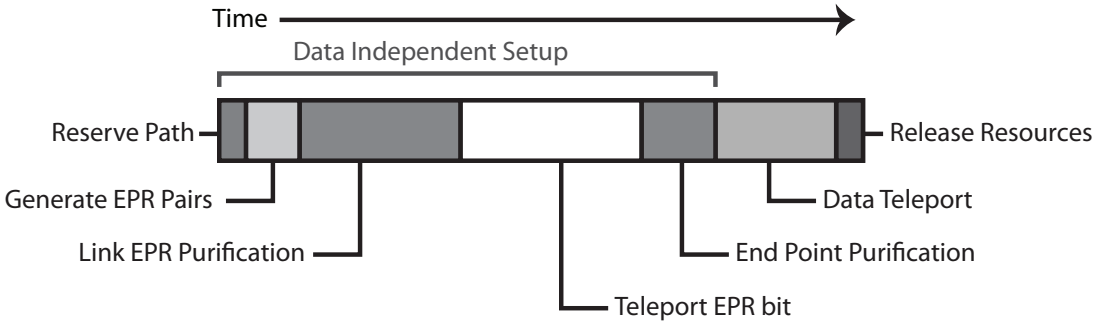



Figure 5.19: Breakdown of time to perform a network connection. All the operations that occur before the Data Teleport stage are considered part of the data independent setup.

### 5.3.6 Connection Breakdown

In Figure 5.19 we see the breakdown of time necessary to establish a network connection. The various pieces correspond to the steps described in Figure 5.15. Of note, a large part of the connection setup process is independent of the data qubit. As shown in the figure, all the steps from Reserve Path to End Point Purification can be done prior to data availability. In Chapter 6, we will exploit the data independence of connection setup in order to optimize the network performance. Our goal will be to preschedule the setup portion so the connection is available for teleportation as close to the time the data completes its operation as possible.

## 5.4 Component Design

The time to set up a network connection and perform a data teleportation is dependent on the design of the various network components. We cannot accurately determine movement latency without macroblock level designs of the routers, purifiers and generators. In this section we provide datapath designs for these components. These components are used during the network synthesis phase of our CAD flow.

### 5.4.1 Purifier

We could implement tree purification (Section 5.2.3) protocol naively at each possible endpoint by including one hardware purifier for each node in the tree (Figure 5.10). For example, if it is known that there will never be a need for more than three rounds of purification at the endpoints, then the tree consists of seven purifiers (four at L0, two at L1, one at L2). So long as this number is low, the entire tree could be implemented in hardware at each possible endpoint. While this method provides minimal latency and maximal pipelining for tree-style purification, the hardware needs quickly become prohibitive as the tree depth increases. Additionally, this mechanism provides no natural means of recovering from a failed purification (that is, the loss of a subtree).

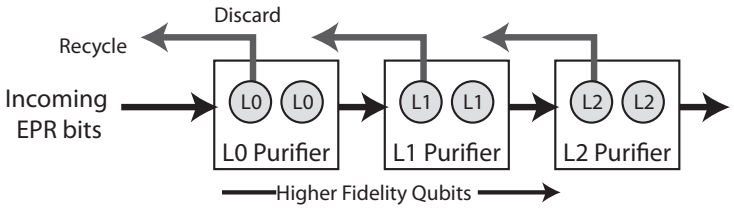



Figure 5.20: Queue purification process. Incoming level 0 qubits are purified in the L0 Purifier. Recycled bits exit and return to the generator units. Purified level 1 bits are then moved into the L1 purifier to await the arrival of additional qubits. This process repeats as necessary.

Instead of designing our purifies in this manner, we use the more robust queue-based purifier shown in Figure 5.20. The incoming stream consists of standard level-0 EPR qubits. If the purifier marked L0 is empty, then the incoming qubit enters it and waits. If it already contains an L0 qubit, the incoming qubit is used to purify the qubit already present. On failure, both are recycled. On success, the new L1 qubit moves into the L1 purifier and waits for a second L1 qubit, or purifies the L1 qubit already there. There are three advantages of this implementation. First, a tree structure of depth $n$ is implemented with $n$ purifiers (rather than $2^n - 1$, as above). Second, movement between levels of purification is minimized, lessening the impact of movement (which is over an order of magnitude worse than two-qubit gate error; see Table 2.1). Third, no special handling for lost subtrees due to failed purification is necessary as they'll be rebuilt naturally.

The primary drawback of this implementation is the latency penalty. If $x$ purifications are needed at level L0, then they must necessarily be done sequentially. This problem may be alleviated by including more queues, however, since each logical communication requires multiple high-fidelity EPR pairs, depending upon the encoding used.

### 5.4.2 Router Links

Figure 5.21 shows the datapath we use to create our network links. The general structure consists of a centrally located EPR generator, connected via ballistic channels to queue purifiers at the end points. Each single link as two ballistic channels. One channel is used to send EPR bits to the purifiers while the other channel returns discarded qubits to the generator for reuse.

At the end points we insert as many queue purifies as are necessary to obtain the appropriate fidelity EPR bit. The purifier design is shown on the right side of the figure.

If more than one link is necessary between neighboring routers we can stack these route link datapaths to obtain the desired bandwidth. In Section 6.5.1 we show how adding more links between routers effects the overall performance of the communication network.

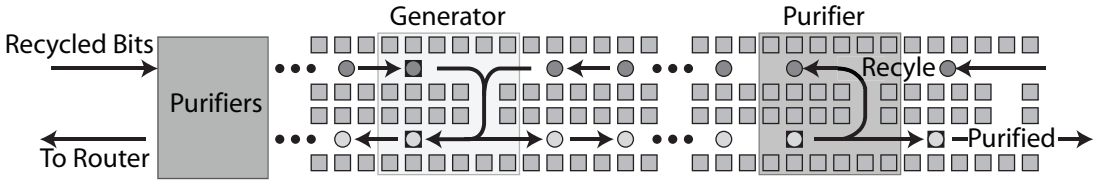



Figure 5.21: Router Link Datapath. Links are composed centrally located generators connected to queue purifiers at the end points. The left side purifiers are not shown for clarity.

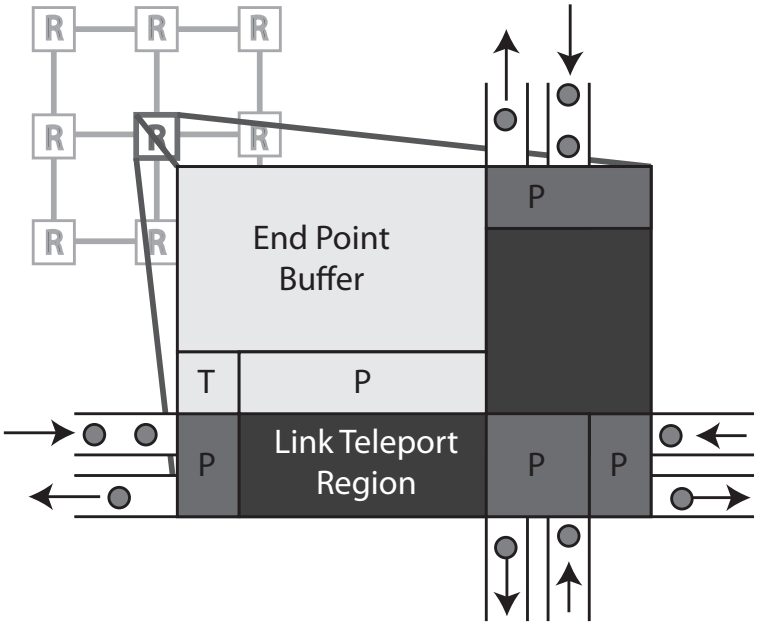



Figure 5.22: Datapath layout for a network router. The link purifiers are shown integrated into the router area. Each network dimension has a bank of link teleporters and a separate set of purifiers and teleporters exist for the end point connections.

### 5.4.3 Routers

The router component in the network both routes multi-link connections and acts as an interface to the network for the local compute region. Figure 5.22 shows the datapath layout for our routers. The purifiers from the router links are shown at the end points of the connected links. These purifies are used to improve the fidelity of the local EPR bits that compose the virtual wire. If the router is acting as a mid-point in a longer chain teleportation the link teleport region is used to store bits until they are ready to teleport to neighboring routers.

If this router is the destination for a connection, the end point purifiers are used for the final purification process and a separate teleporter is used to conduct the data teleport. We insert buffer space within the router to temporarily hold the data qubits

until the connection is ready for use. Once the data qubits have teleported in, they exit the router to the local compute region. Our tool set uses the short-distance communication scheduler( 4.2.5) to schedule all the movement within the network components and the movement between the router and the compute regions. This overhead is appended to the raw data teleport time and inserted into the overall communication time.

# Chapter 6

# Routing

The communication network portion of the quantum datapath is responsible for establishing connections between the numerous compute regions. In Section 5.3 we described the various phases of a network communication operation in detail. As we previously mentioned only a small portion of the total connection time is dependent on the data qubits that will be moved and therefore a large part of the communication setup and preparation can occur before the data is ready to use the connection. By completing the data-independent phase of the network connections early we can minimize the overall delays introduced by the long-distance communication operations.

Establishing a network connection can be decomposed into two major parts:

**Routing:** A connection is constructed by establishing a path through the network over which all the EPR pairs will be linked. This path must be routed through the network in order to determine the resources that will be allocated to the connection. If the desired resources are not available at the scheduled times the connection will stall until they become available.

**Scheduling:** Each connection must be scheduled to start at a given time. The scheduled start time determines when the connection reserves its needed resources and ultimately determines when the connection will be ready for use.

In a perfect scenario all the network connections are scheduled such that they have completed their setup and preparation and are ready for final data teleport at the exact moment the data qubits are ready to move. A network connection that isn't ready in time causes the data qubits to sit idle, potentially introducing errors resulting from decoherence. In fact, as we show in Section 7.1, the difference between pre-scheduled and on-demand connection setup can make the difference between a functioning and non-functioning circuit. Further, network connections that are ready well before the data qubit is ready unnecessarily consume network resources as the connection must remain active until the final data teleport occurs.

In this chapter our goal is to efficiently use available network resources and reduce overall circuit latency. We will discuss techniques to route connections within the network so that they are ready as close to the time when data is ready to teleport as possible.

## 6.1 Circuit Schedule

The input to the routing phases consists of of a set of events produced by the mapping stage in the CAD flow. These events are determined by mapping all the gate operations to specific functional units within a compute region and listing all the movement that must occur to get a qubit to it's assigned functional unit. An event can be one of three types: gate operation event, local ballistic move event, or long-distance move event.

As an example consider the datapath shown in Figures 6.1, which contains four compute regions. In this example, each compute region contains two functional units – each of which holds sufficient resources to perform a two-qubit gate. Figure 6.2 shows a possible mapping of a dataflow graph to the example datapath. Each vertex is assigned to a compute region and the edges in the graph that correspond to long-distance moves are highlighted in bold. An example mapping for this circuit resembles the following:

```
[   0] GateEvent:  H Q0     @ CR2:FU0
[   0] GateEvent: CX Q2, Q3 @ CR1:FU0
[  10] MoveEvent: Q0 -> CR0:FU0 [network]
[ 100] GateEvent:  H Q3     @ CR1:FU0
[ 100] MoveEvent: Q2 -> CR2:FU1 [network]
[1200] GateEvent: CX Q0, Q1 @ CR0:FU0
[1300] MoveEvent: Q1 -> CR2:FU1 [network]
[2400] GateEvent: CR Q1, Q2 @ CR2:FU1
```

The routing phase takes this event list and routes all the network connections assigning them to the appropriate links and updating the expected start times for all the events. In this example, if the $Q0$ move from $CR2$ to $CR0$ needs to be delayed, all the subsequent operations are delayed to maintain data dependencies.

A fully specified circuit schedule produced by the routing phase must contain a route and scheduled start time for all long-distance communication connections. As an example, consider the data flow graph and network shown in Figure 6.3. In this example, there are three communication edges (the bold lines) that require the network. Each of these connections must have a corresponding path and setup start time in the datapath's network. A connection reserves resources beginning at the setup start time, sets up the communication channel and then is available for data teleportation.

To state the problem more formally, given the circuit dataflow graph $G = (V, E)$ where $V$ is the set of gate operations that occur and $E$ is the set of communication edges between gate operations (some local, some long-distance), the network $N = (R, L)$ where $R$ is the set of routers in the network and $L$ is the set of EPR distribution links between routers, and a list of connections $C$, a valid schedule maps each connection in $C$ to a set of routers $R$ and links $L$ and indicates the start and end time for the connection.

The schedule must respect the data ordering specified in $G$ such that if edges exist from $v_1$ to $v_2$, any connection used to get data to $v_2$ must be scheduled during or after all connections used to get data to $v_1$. In Figure 6.3, the connection for $(v2, v4)$ must occur during or after the connection for $(v0, v2)$ since it depends on data generated by

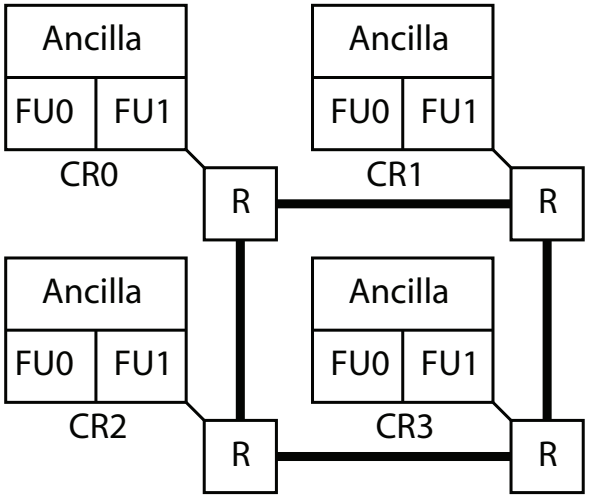



Figure 6.1: Example 4 node datapath. This datapath contains four compute regions with their corresponding routers. Each compute region (CR) contains ancilla generators and two functional units. The thicker lines connecting routers are the network links.

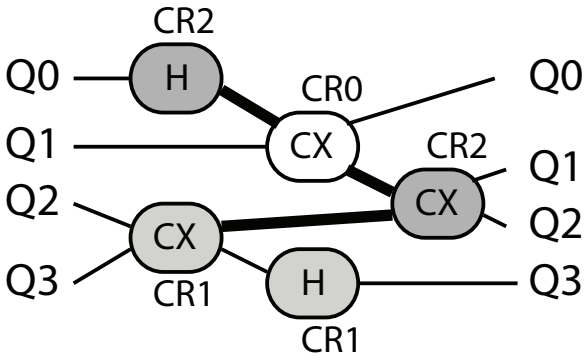



Figure 6.2: Example four qubit 5 gate dataflow graph. Each vertex is assigned to execute within a specific compute region. Solid edges between vertices represent long-distance network connections. All other edges are ballistic movement within a compute region.

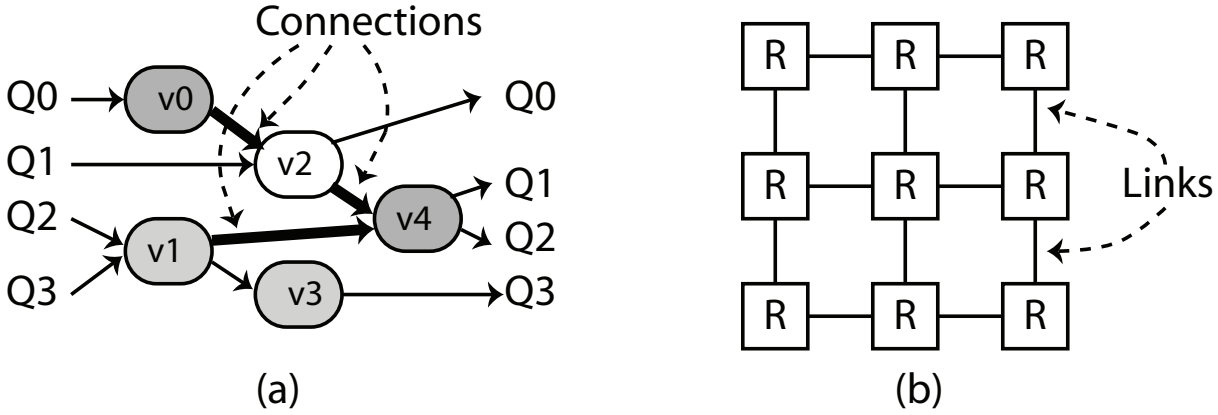



Figure 6.3: Circuit and Network model used to create a schedule of communication. In (a), the bold lines are communication edges that require the use of the network.

$v2$. If a later connection reserves resources before the previous connection is established the schedule may result in deadlock. Additionally, two connections that share a link in the network must be ordered such that one completes before the other uses the link. We chose not to share links between connections since sharing the resources during an active connection would delay the overall connection. Our goal is to minimize the amount of time qubits idle. Longer connection setup times (due to sharing resources) would increase the idle time of the EPR bits and potentially the data qubits.

## 6.2 Routing

Routing can either be performed dynamically (i.e. on-line, utilizing local information to route messages through the network) or statically (i.e. off-line, utilizing static analysis of the complete quantum circuit to pick routes through the network). Obviously, the static option has the greatest potential for optimization, but may also consume an unreasonable amount of computational resources in order to produce an optimal result.

One obvious option for routing and scheduling would be the equivalent of a classical multiprocessor network, one that operated dynamically by opening connections on-demand at the time that a quantum bit is ready to transmit. In this section, we will introduce our routing variants in the context of on-demand routing. Later, in the static scheduling algorithms of Sections 6.3 and 6.4, routing will be called as a subroutine.

The routing portion of the network connection setup takes the source and destination points for the connection and selects a consecutive set of routers to establish the connection. The routers then reserve resources for the connection and perform all the necessary EPR purification and EPR teleportation with the ultimate goal of distributing the appropriate number of EPR bits for use by the data qubit.

### 6.2.1 Dimension Order

The first routing technique we will discuss is Dimension Order, a simple algorithm widely used within classical networking. Dimension Order routing makes for an excellent reference point to evaluate our more complex adaptive algorithms. Further, it is provably deadlock free.

In our implementation the long-distance communication network is constructed as a two-dimensional mesh grid with each router assigned a coordinate value given its position in the grid. The path from source router to destination router is constructed by first traversing the x direction until reaching the corresponding x value of the destination router, followed by doing the same in the y dimension until the destination router is reached. Algorithm 1 shows pseudo-code for Dimension Order Routing.

Dimension Order routing is simple and easy to implement but it does come with some drawbacks in that links can quickly become oversubscribed. If too many connections are requested between the same source and destination router all the connections will attempt to use the same path through the network and will result in significant network delays. To combat this problem we consider Adaptive routing algorithms.

**Algorithm 1** Dimension Order Routing

```
(src.x, src.y) ⇐ Source Router coordinates
(dst.x, dst.y) ⇐ Destination Router coordinates
path ⇐ [src]
while src.x ≠ dst.x do
  if src.x > dst.x then
    src.x ⇐ src.x − 1
  else
    src.x ⇐ src.x + 1
  end if
  path.APPEND(src)
end while
while src.y ≠ dst.y do
  if src.y > dst.y then
    src.y ⇐ src.y − 1
  else
    src.y ⇐ src.y + 1
  end if
  path.APPEND(src)
end while
return path
```

### 6.2.2 Adaptive

We can improve upon the Dimension Order router by using adaptive routing techniques. Adaptive routing picks a path through the network given the network state with the goal of distributing connections across the available network resources to minimize network delays. The two variations of adaptive algorithms we study are *Minimal Adaptive* and *Full Adaptive*.

Minimal Adaptive picks the best path from a source router to a destination router with a maximum length equal to the length of a Dimension Order path. This approach allows the network to balance the load across a number of minimum distance paths between the end points.

In contrast, Full Adaptive picks the best path between source and destination but unlike Minimal, it allows paths longer than the Dimension Order path. This addition allows us to use much longer paths in our efforts to minimize the amount of delay introduced by the network.

There are a number of pros and cons to each of these Adaptive schemes. Minimal Adaptive has a much smaller set of potential paths to choose from and limits the control complexity required to pick a path. However, because of the limited choices it cannot take advantage of unused network resources outside of the selection area. Full Adaptive routing allows us to search throughout the whole network to find the best path. Unfortunately as network sizes increase the number of potential paths between any given source

and destination increases dramatically. This larger search space complicates control and selection of the paths. Additionally, edge cases exist making it possible that a circuitous path chosen for one move may end up consuming resources best saved for a future move thereby delaying the overall circuit latency.

The high-level algorithm for Adaptive routing is shown in Algorithm 2. FindPaths determines all the paths between the source and destination router up to a given maximum length. The only difference between Minimal and Full Adaptive is the value used for $max\_length$. For Minimal $max\_length$ is set to the length of the Dimension Order path. In the ideal case, Full Adaptive would set $max\_length$ to $\infty$ in order to search for all possible paths between the $src$ and $dst$. Unfortunately, this procedure would be very expensive for larger network sizes. Instead we limit the search to paths that are the length of the DO path plus six, allowing two turns "away" from the destination to avoid congestion while minimizing the path search space. Further, we do not allow cycles in the path.

**Algorithm 2** Adaptive Routing

$(src.x, src.y) \Leftarrow$ Source Router coordinates
$(dst.x, dst.y) \Leftarrow$ Destination Router coordinates
potential_paths $\Leftarrow$ FindPaths$(src, dst, max_length)$
$path \Leftarrow$ BestPath$(potential_paths)$
**return** $path$

Once we have a list of all possible paths to get from source to destination we move on to selecting the best one among them which is performed in the call to BestPath. BestPath ranks the paths according the three criteria in this order:

1. **Teleport Ready Time** The time the teleport will be ready to run for the requested move. If there is congestion along a given path the teleport will be delayed until the path frees up. We determine which paths have the earliest ready time and remove all the other paths from consideration.

2. **Spare Capacity** For each link in the path we determine how much spare capacity will remain in the link after this connection is created. The goal is to select the path that maximizes the remaining capacity, biasing the paths to lower utilized paths making it more likely future connections will not be delayed.

3. **Path Length** The total length of the path using by the connection. For Minimal Adaptive this value will be the same for all paths under consideration. For Full Adaptive we use the shorter length paths to minimize resource utilization in the network.

## 6.3 Optimal Routing and Scheduling

In this section, we start our exploration of off-line scheduling by developing a 'guaranteed not to exceed' optimal routing schedule. Although our solution will be impractical

for large circuits, it will give us a minimum baseline latency to compare for small circuits. Further, by formulating the optimal solution, we will gain insight into what is required for a heuristic solution, as discussed in the following section.

We can determine an optimal schedule and routing for a quantum circuit on a given network by framing the problem as a Mixed Integer Linear Program (MILP), a technique frequently used to solve scheduling problems. One approach to creating a MILP is to extend a standard network-flow Linear Program (LP) to account for time. A single connection in our teleportation network can be considered a network flow and the LP can solve for link usage at every time unit. This formulation results in a single LP to generate full routing and scheduling for all connections. Unfortunately such a program quickly becomes too large to be feasible.

Instead of attempting to solve the full program in a single pass, we break the problem into smaller subproblem. Our solution iteratively solves an LP until the optimal schedule and routing is obtained. The goal is to continuously limit the search space of subsequent iterations by adding new constraints obtained from the current iteration.

### 6.3.1 MILP for Quantum Routing

The iterative algorithm we use is shown in Algorithm 3. The input to the algorithm consists of the following: the datapath network $N(R, L)$ where $R$ is the set of routers in the network and $L$ is the set of links connecting the routers, the quantum circuit dataflow graph $G = (V, E)$ where $V$ is the set of quantum operations and $E$ is the set of communication edges between the operations, and a set of connections $C$ which enumerates all the long-distance communication operations with their source and destination routers. Each connection $c \in C$ corresponds to an edge in $E$, however not all edges will have a connection as movements within a compute region do not require a long-distance connection.

We initialize the algorithm with the base LP variables and constraints described here:

#### Variables

Each connection is created as a path of links through the network. $x_l(l, c)$ indicates whether or not a connection uses a given link in the network.

$$\forall l \in L, \forall c \in C$$

$$x_l(l, c) = \begin{cases} 1 & \text{if } c \text{ uses link } l \\ 0 & \text{otherwise} \end{cases}$$

All the connections must be ordered in time if they share a link. This way we know which connection is established first. The subsequent connections must wait for the prior ones to complete before the resources are allocated. $x_o(c_1, c_2)$ tells us if $c_1$ precedes $c_2$ in the program run.

$$\forall c_1, c_2 \in C$$

**Algorithm 3** Linear Program Optimization

```
latency = ∞
constraints = InitialLinearProgram(G)
AddOrderingConstraints(constraints, G)
while true do
  schedule = LPSolve(constraints)
  if schedule = Unsolvable then
    return latency
  end if
  UpdateGraph(G, schedule)
  if G has cycles then
    AddCycleConstraints(constraints, G)
  else
    latency = min(latency, ComputeLatency(G))
    constraints = AddCriticalPathConstraint(constraints, G)
  end if
end while
return schedule
```

$$x_o(c_1, c_2) = \begin{cases} 1 & \text{if } c_1 \text{ precedes } c_2 \\ 0 & \text{otherwise} \end{cases}$$

### Constraints

If a connection is set to occur before another connection, it must not also occur after the connection. Both $x_o(c_1, c_2)$ and $x_o(c_2, c_1)$ cannot be 1.

$$\forall c_1, c_2 \in C, c_1 \neq c_2$$

$$x_o(c_1, c_2) + x_o(c_2, c_1) <= 1$$

A connection cannot occur before itself.

$$\forall c_1 \in C$$

$$x_o(c_1, c_1) = 0$$

If two connections share a link, they must also specify an ordering.

$$\forall c_1, c_2 \in C, c_1 \neq c_2, \forall l \in L$$

$$x_o(c_1, c_2) + x_o(c_2, c_1) >= x_l(l, c_1) + x_l(l, c_2) - 1$$

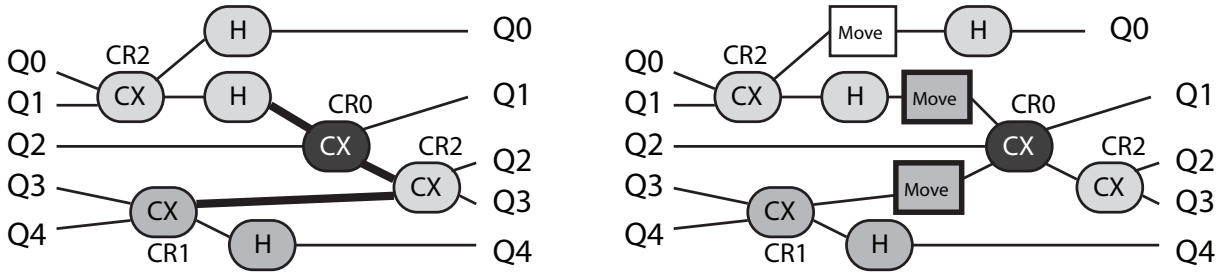



Figure 6.4: Adding move event edges to the dataflow graph.

### Objective

Minimize the number of orderings:

$$\text{minimize} \sum_{c_1, c_2 \in C} x_o(c_1, c_2)$$

### Ordering Constraints

The linear program variables only deal with connections in the network. To generate a correct schedule of communication we need to make sure these connections don't violate the dataflow graph dependencies. In the input dataflow graph, each node represents a quantum operation, and the edges are communication between the operations. Some edges require long-distance communication, while others are only ballistic movement within a compute region.

We take this graph and transform all network connection edges into move operation nodes. This process is illustrated in Figure 6.4. The bold edges are the network edges, which eventually are converted into `MOVE` nodes. Once we have this graph, we can quickly search through it to generate all move ordering constraints. If a path exists from a $move_a$ to $move_b$ we add an ordering constraint to the linear program formulation asserting that $move_b$ must either occur at the same time or after $move_a$.

### Output

The output of the LP optimization are the variables indicating which links a connection uses, and what order the connections occur (if any links are shared). We process this information and update the transformed dataflow graph as shown in Figure 6.5. When two move operations are ordered an edge is added between them. Using the new graph we first check to see if any cycles exist. If a cycle exists the move operations that formed the cycle are added to the LP constraints and the LP process is repeated.

If the output doesn't contain any cycles, we can safely calculate the critical path and determine the total latency of the circuit. If the latency improves on the prior best value we save the schedule. A constraint is then added to the LP to prevent the same critical path from occurring again and the LP process is repeated. When an LP runs indicates no feasible solution, we output the best schedule encountered.

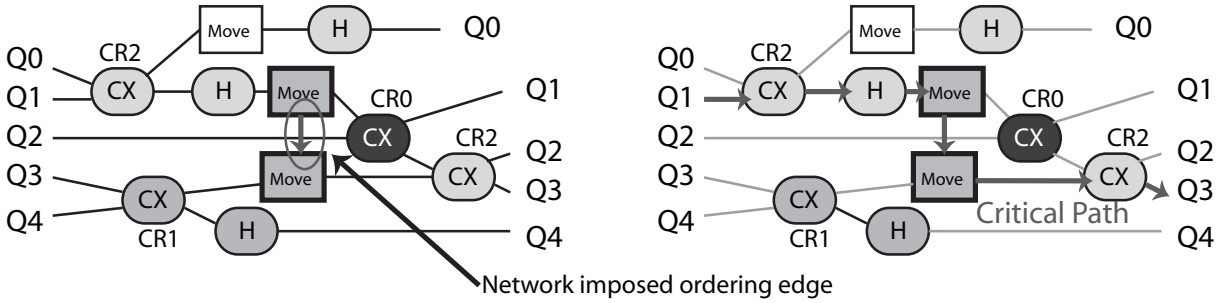



Figure 6.5: Adding network ordering edges to the graph.

### 6.3.2 Results

We use the GNU Linear Programming Kit (GLPK) [26] to solve our linear program formulation. We formulate the problem in the GNU MathProg modeling language and execute the command line version of GLPK to solve the problem. Our tools post process the output to recreate the solution in our internal representation. As we progress through the search for an optimal solution we simply append new constraints onto the base LP description and rerun GLPK.

The time necessary to properly determine an optimal solution using the LP method increases dramatically as the problem size increases. As an example, solving a circuit with 90 gates on a $3 \times 2$ node network can take anywhere from 90 minutes to multiple days to search all critical paths (on a 2 Quad-Core Intel Xeon running at 2.67GHz). The matrix generated is on the order of 10000 rows by 5000 columns and requires many iterations to search the full space.

Even though we made a number of optimizations to decrease the number of iterations necessary to solve the LP, once we enter circuits with 1000s of gates and networks bigger than $4 \times 4$ the LP technique is infeasible. Unfortunately, given the 10 day run time limit imposed on us by our compute resources, the LP was unable to solve a single iteration of the problem. An optimal solution to the LP was found, however the integer optimization stage of the calculation failed to find an integer solution.

## 6.4 Practical Scheduling and Routing

While it is interesting to determine an optimal solution to the scheduling and routing problem for small circuits, we are most interested in evaluating larger circuits. As we mentioned above, the optimal search cannot be performed on circuits of these sizes. Instead, we introduce a set of heuristic algorithms and will show that our heuristics generate solutions within 10% of optimal and do so in dramatically less time, allowing us to study larger circuits (Chapter 7).

### 6.4.1 On Demand Scheduling

The base-line scheduling technique we present is the On Demand method. In this method the network determines a route and initiates the connection starting at the time a communication request is encountered. Since the network requires time to properly establish the connection and distribute all the EPR bits the teleport operation must stall until the connection is ready.

The On Demand method will be the base-case to which we will compare our alternate techniques. In our implementation of On Demand we use a Minimal Adaptive routing scheme to generate the paths. The basic algorithm used is presented in Algorithm 4.

**Algorithm 4** On Demand Scheduling

```
schedule = MAP(G, datapath)
schedule_routed = ∅
while schedule not empty do
  event = NEXTEVENT(schedule)
  t_start = DETERMINEONDEMANDSTART(event)
  ROUTEEVENT(event, t_start)
  schedule_routed.APPEND(event)
  UPDATEDEPENDENCIES(event)
end while
return schedule_routed
```

The major functions that compose the algorithm are:

MAP($G, datapath$): The mapper phase maps each operation in the circuit graph to a compute region in the datapath. Each operation is assigned a start time to indicate the correct ordering of operations within the compute regions. The mapper inserts communication operations into the schedule of operations whenever a qubit must move from one compute region to another.

NEXTEVENT($schedule$): The algorithm processes events in the order determined by the mapping phase. The first ready event is removed from the schedule and processed.

DETERMINEONDEMANDSTART($event$): This function determines the start time for the event given dependencies and network congestion. If the event is a gate operation, it is set to start when all the qubits will arrive in the compute region. For communication events the start time is dictated by when the previous gate that uses the qubit completes.

ROUTEEVENT($event, t_{start}$): This function determines the full route the connection will take within the network. It uses the minimal adaptive routing algorithm described in Section 6.2 and only returns connections that will start the setup procedure at $t_{start}$. If no route exists that can start at $t_{start}$ the connection is delayed until the first available route can be used, updating the connection to reflect the delay.

UpdateDependencies(*event*): After an event is scheduled we update dependency tracking tables so future events that depend on the current event's resources are appropriately delayed.

### 6.4.2 Heuristic Prescheduling

As we previously mentioned one of the major benefits of using a teleportation based long-distance communication network is the ability to remove significant portions of the communication setup times from the critical path. This optimization is performed by prescheduling the communication setup so it will complete just before the data qubit needs to move. At compile time we have global knowledge of the circuit run and we utilize this information to schedule the move operations prior to run-time.

**Algorithm 5** Heuristic Prescheduling

```
schedule = Map(G, datapath)
schedule_routed = ∅
while schedule not empty do
  event = NextEvent(schedule)
  t_start = DeterminePreScheduledStart(event)
  RouteEventPreScheduled(event, t_start)
  schedule_routed.append(event)
  UpdateDependencies(event)
end while
return schedule_routed
```

Algorithm 5 outlines the prescheduling algorithm. The methods used are identical to the On Demand Scheduling algorithm with the exception of:

DeterminePreScheduledStart(*event*): Unlike the on demand case, the prescheduling algorithm tries to schedule connections before the data bit is ready to use the connection. When a movement event is encountered, this function returns the earliest time a data qubit is ready to move to indicate that the connection must be complete and ready to use by this time. The goal is to perform the final data teleport at the exact moment the data completes it's previous operation.

RouteEventPreScheduled($event, t_{start}$): The prescheduled version of RouteEvent attempts to find a valid route for the connection that will complete and be ready for the data teleport to start at $t_{start}$. This function tries to find a route where all the setup overhead can be done before the data qubit is ready to teleport. The process uses the same routing algorithms as RouteEvent but must also calculate the connection setup start time. In the on demand case, the connection setup start time is $t_{start}$. For prescheduling, we start by assuming the connection will start at $t_{start} - t_{dopath}$, the requested time minus the time to use a dimension ordered path. We search for all the routes that are possible at that time. If our adaptive routing

returns a path longer than the dimension ordered path we have to check to see if the route is still valid with the connection setup start time set to $t_{start} - t_{path}$. If valid, the route is returned. If not we repeat the process with the next best route in the list.

### 6.4.3 Simulated Annealing

The final algorithm that we propose is on Simulated Annealing (SA). The SA algorithm is shown in Algorithm 6. The general structure of our algorithm follows that of most other SA implementations. We start with a high temperature and make random moves within our search space. All moves that lead to a better schedule are accepted, and depending on temperature moves that increase total latency may also be accepted with some probability. Eventually we reach a target temperature and the best schedule obtained so far is returned. In searching for an optimal solution, we will allow a fully adaptive solution.

The initial schedule is obtained by running our heuristic prescheduling algorithm with full adaptive routing and with minimal adaptive routing. The best schedule returned is used to start the annealing process. For each iteration, we pick a random connection from the list of network connections and modify it as indicated in the description of MOVE. All moves that reduce the circuit latency are accepted, and a move that increases the circuit latency is accepted with some probability. The goal of simulated annealing is to try to move out of any local minima that the greedy heuristic approach may have fallen into. By randomly changing a connection route and start time, we may jump out of the local minima and determine a better overall circuit schedule.

**Algorithm 6** Simulated Annealing Search

```
latency_best, schedule_best = min(FULLADAPTIVE(G, N), MINADAPTIVE(G, N))
latency = latency_best, schedule = schedule_best
cost = COST(schedule), temp = T_0
while temp < T_end do
  schedule_new = MOVE(schedule)
  cost_new = COST(schedule)
  if ACCEPTTRANSITION(temp, cost, cost_new) then
    cost = cost_new
    schedule = schedule_new
    if cost < cost_best then
      cost_best = cost
      schedule_best = schedule
    end if
  end if
  temp = TEMP(temp)
end while
return schedule_best
```

For our implementation we define the major components of the algorithm as follows:

COST($s$): The cost function returns the quality of the passed in schedule. In our case COST($s$) returns the total latency of the schedule.

TEMP($t$): The temperature function calculates a new temperature given the current temperature and the iteration number. We transition the temperature only after a set number of iterations (determined by the circuit size).

ACCEPTTRANSITION($t, c, c_{new}$): This function determines if a given transition in the schedule will be accepted given the current temperature. A transition is allowed if:

$$e^{-\frac{c_{new}-c}{t}} >= RandomDouble(0,1)$$

MOVE(*schedule*): Move makes a single change to the current schedule's routing or connection ordering and returns a new schedule with the change applied. From the current schedule we pick a random connection and with equal probability decide if we will change the route of that connection or the ordering of that connection. If we decide to change the route of the connection we generate a random route through the network for the connection, apply it and return the new schedule. If we instead decide to change the ordering of connections we add a connection ordering edge in the schedule graph making sure to avoid any cycles (two connections sharing a link must be ordered to match the circuit dataflow graph).

## 6.5 Routing Analysis

The main goal of this work is to present a communication infrastructure for a quantum datapath and show how we can optimize it to best run a given quantum circuit. The naive approach to establishing connections is to execute the quantum circuit and create connections on demand. This approach is simple to implement but leaves a lot of room for improvement.

With the various routing and scheduling techniques in hand we now analyze their performance under various conditions. Since there are multiple choices for both the routing and the scheduling technique used we've chosen to narrow our search space to the combinations listed in Table 6.1. From here on out we will refer to a combination of routing and scheduling by using the type name listed in the table.

Ideally we would like to use the linear program technique to generate the optimal schedule and routing for all the quantum circuits we would like to evaluate. Unfortunately, running an MILP for interesting sized circuits is impossible. Instead we've presented a heuristic approach that comes very close to the optimal schedule but does so in a reasonable amount of time (allowing us to study the large circuits described in the next Chapter).

We are mainly going to rely on our heuristic routing and scheduling approach so we start by comparing our approach to the optimal case. Using small randomly generated

| Name | Scheduling | Routing |
|---|---|---|
| Optimal | Linear Program | Full Adaptive |
| On Demand | On Demand | Minimal Adaptive |
| Dimension Ordered | Heuristic Prescheduling | Dimension Ordered |
| Minimal Adaptive | Heuristic Prescheduling | Minimal Adaptive |
| Full Adaptive | Heuristic Prescheduling | Full Adaptive |
| Simulated Annealing | Simulated Annealing | Full Adaptive |

Table 6.1: Routing and Scheduling Types

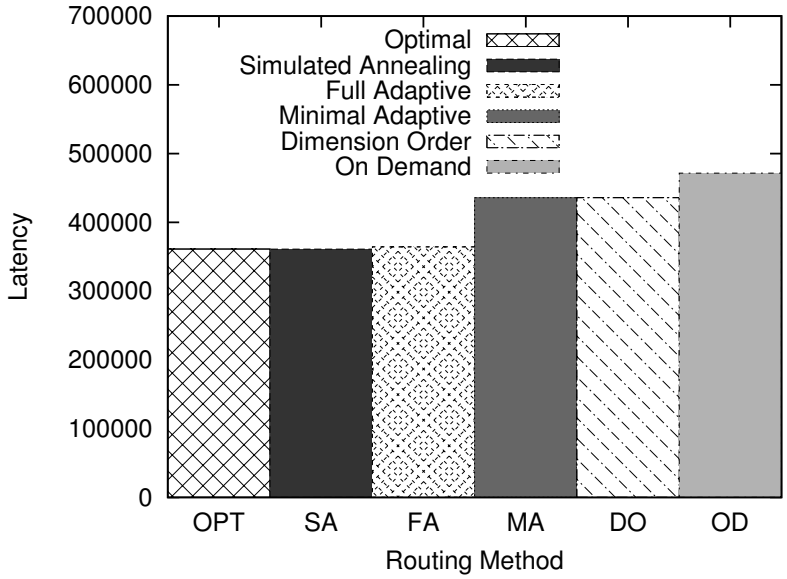



Figure 6.6: Circuit Latency for different Routing techniques. For small circuits the Simulated Annealing and Full Adaptive routing techniques come very close to the optimal time as determined by the Linear Program.

graphs, we ran our CAD flow using the various routing schemes described above. A representative sample of these results are shown in Figure 6.6. The main take away is that our adaptive techniques come very close to the optimal schedule generated by the linear program and does so in considerably less time (minutes to hours compared to multiple days). Consequently, from here on out we won't present additional results from the LP or SA algorithms and instead mainly rely on the results obtained from our heuristic approach.

### 6.5.1 Network Area Impact

The main factor that determines the performance of the network is the amount of area and resources dedicated to the network. We start by studying the effects of routing technique on overall circuit latency. Figure 6.7 graphs the performance of a random circuit

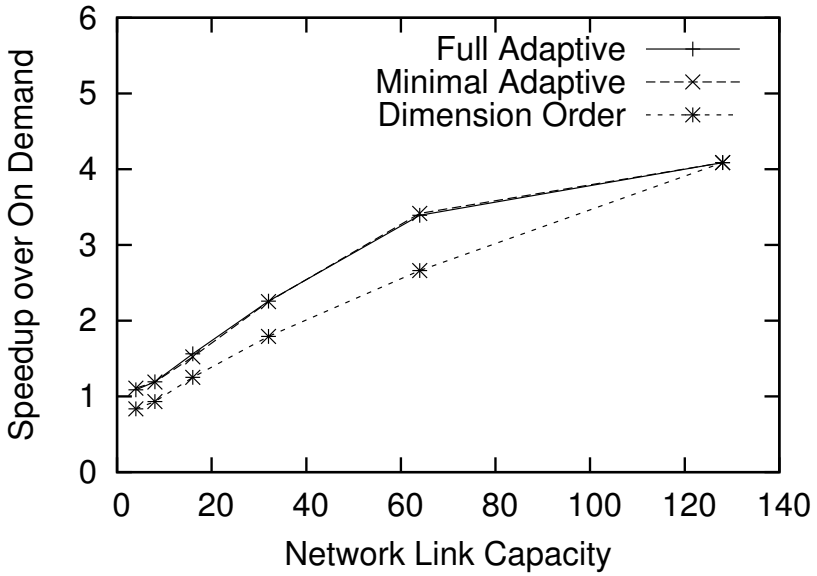



Figure 6.7: Network speedup versus link capacity. Each point is normalized to the circuit latency obtained by using On Demand routing. As link capacity is added to the network the adaptive algorithms have more options for routing around congestion and improve over the Dimension Order routing. At the largest network size Dimension Order routing catches up as it no longer encounters congestion.

given increasing network link sizes. These results were generated using an 8x8 mesh grid where the only parameter varied is the width of the links connecting the routers. Each point in the graph is plotted as the speedup over the on demand baseline result.

As demonstrated in the graph, the speedup we can achieve is directly linked to the size of the network provided in the datapath. For low link capacities, our optimized routing and scheduling yields almost no improvement over the base case, regardless of which routing technique we utilize. When the network is sized this small, there is no opportunity to preschedule any connections as the network is fully utilized. In this case, almost every communication operation contributes delay to the overall circuit latency.

As we increase the link size we start to see significant improvement in total circuit latency when compared to the on demand baseline. Both of the adaptive routing techniques perform almost identically and are able to outperform the dimension ordered case. This result is expected since the dimension ordered case is unable to compensate for any congestion (all connections between two given routers will always take the same path). Overall, we are able to achieve a speedup of a factor of 4 for larger link capacities.

Figure 6.8 plots the average amount of delay introduced by long-distance communication. The delay is the amount of time between when a connection is requested, and when the data teleport is ready. Congestion within the network delays connections until a link is available. For the on-demand scheduler, the delay will always include the connection setup time (in addition to any network congestion). In contrast, the heuristic scheduler tries to preschedule connections to remove any connection setup delay.

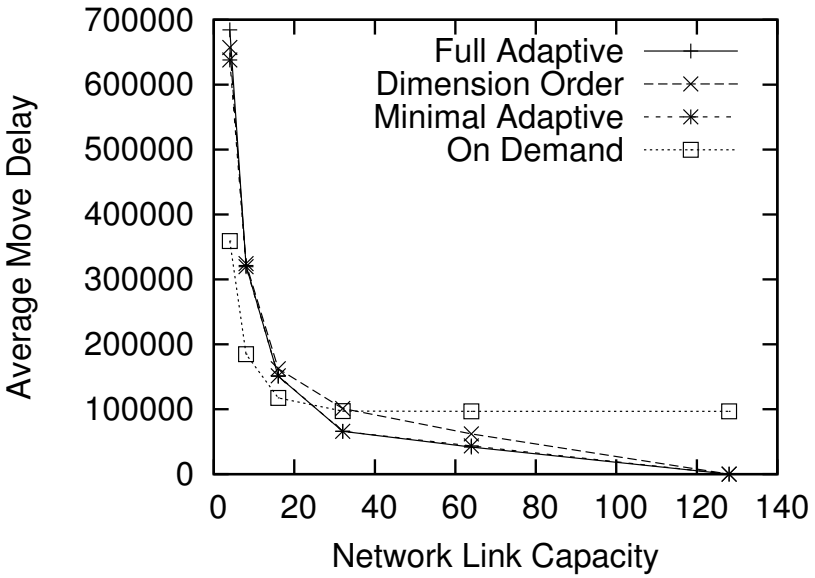



Figure 6.8: Average per move delay introduced by the network. For each routing technique, the average amount of delay to move a qubit through the network is plotted against the size of the network links.

In the figure we see all the scheduler and routing types have high delay resulting from an oversubscribed network. This effect is further illustrated in Figure 6.9 where we see the fraction of move operations that are delayed. For small link capacities, almost 80% of move requests are delayed due to congestion. There is no line for the on-demand case because all connections are delayed in the on-demand case since we consider connection setup time to be part of delay.

As the size of the network increases, the average move delay decreases accordingly. Eventually the heuristic prescheduled techniques reach a point where no delay is introduced by network connections. This value is the ideal point where all connections can be prescheduled in time for the data teleport to occur. The on-demand scheduler asymptotes at a value above zero due to the lack of prescheduling.

### 6.5.2 Sensitivity to Mapping

As we mentioned earlier in this chapter our router operates on a list of mapped instructions as generated by the high-level coarse mapping phase of our CAD flow. The mapper decides where instructions are run in the datapath. The router is bound by the decisions made by the mapper.

Figure 6.10 illustrates how the routing techniques perform using different mappers. We generate a random circuit and then use a unintelligent randomized mapper to create the mapping of instruction to compute region. A number of random mappings are tested and the average is plotted in the graph with minimum and maximum error bars.

As we can see in the graph, our routing techniques are able to generate a speedup

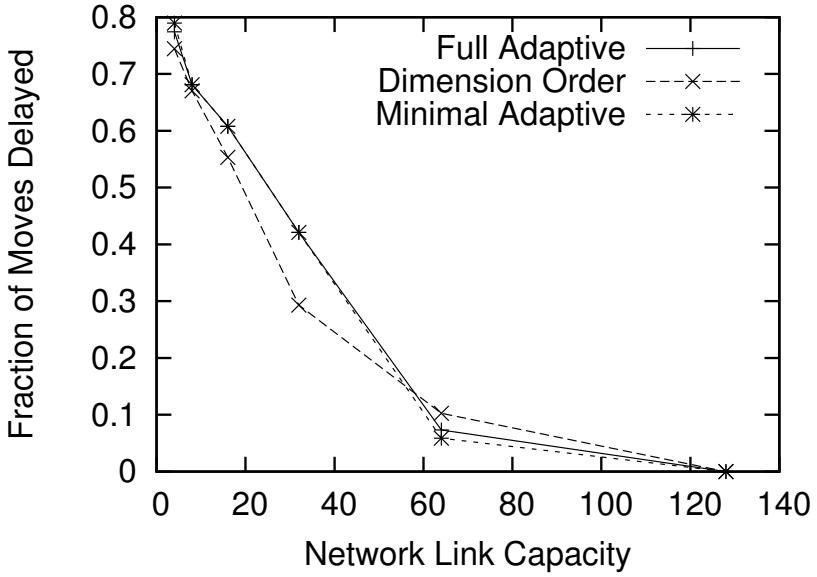



Figure 6.9: Fraction of total network connections that are delayed. When a connection encounters congestion it must be delayed until the desired path frees up. As more link capacity is added the number of moves delayed decreases. Dimension Order has fewer delays initially because the communication requests are spread out resulting in a longer total latency when compared to the Adaptive schemes.

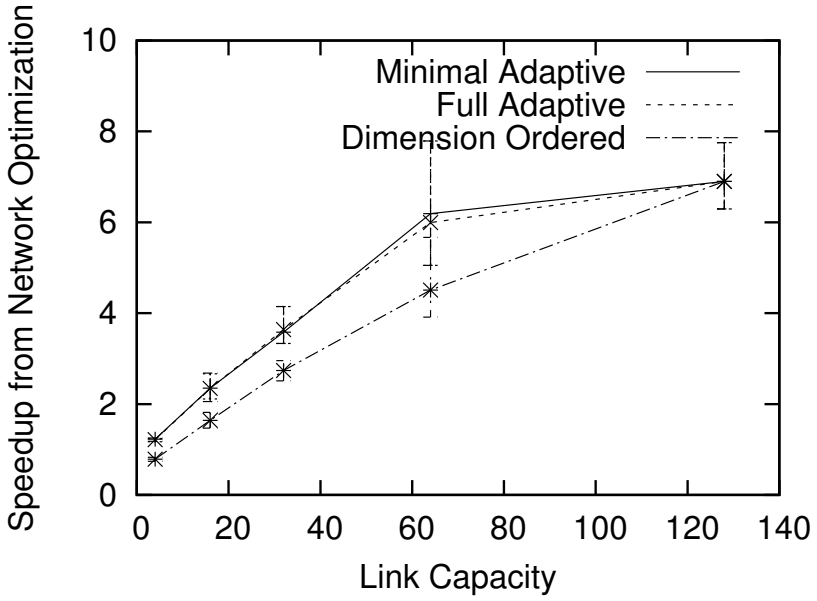



Figure 6.10: Network speedup sensitivity to Mapping technique. The circuit is mapped using multiple random Mappers and then routed using the different techniques. For each data point the average speedup is plotted along with the minimum and maximum speedup (over On Demand routing).

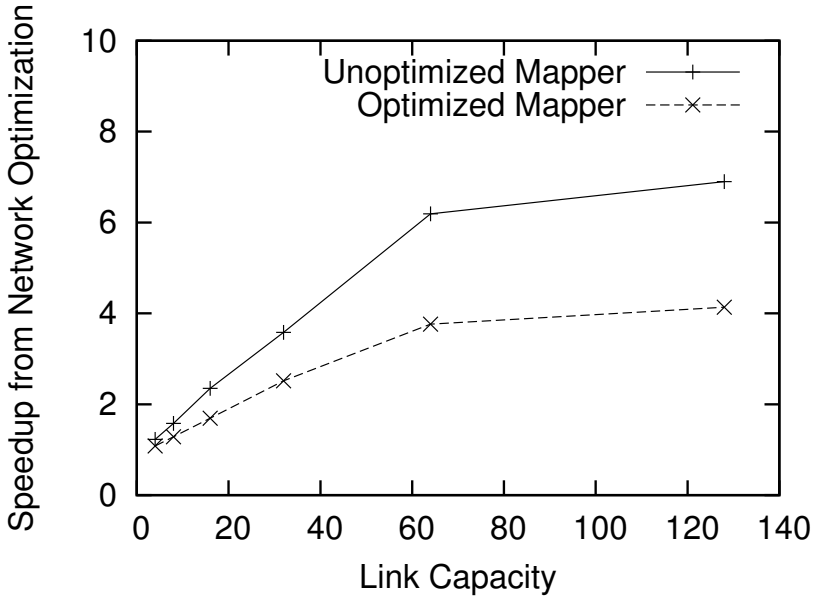



Figure 6.11: Network speedup over on-demand given optimized and unoptimized Mappers. The unoptimized mapper generates a poor mapping with less locality allowing the optimized routing techniques to obtain better speedup. The optimized mapper balances computation more effectively reducing speedup (over On Demand routing).

ranging from a factor of .5 (at small network sizes) to 6 (at larger network sizes). One might ask why optimized routing is able to extract up to a factor of 6 speedup for a random mapper when in the previous Section we only demonstrated a maximum speedup of a factor of 4. This difference is explained by the communication pattern created by the mapper.

The intelligent mapper we use within our CAD flow exploits communication locality and attempts to map neighboring instructions to the same compute region. Additionally, if an instruction must be scheduled into a different compute region the intelligent mapper picks a destination to minimize connection setup.

When compared to our intelligent mapper, the random mappings end up generating a lot more network communication, as no communication locality is used to generate the mapping. The locality differences causes the unoptimized router to delay the circuit considerably while the network connections are made whereas the optimized router is able to preschedule connections more efficiently and therefore generates a much greater speedup.

We compare our optimized router given optimized and unoptimized mappers in Figure 6.11. A single circuit is run on datapaths with varying link capacity. Here you can see the optimized router is able to extract up to a factor of 6 speedup given unoptimized mapping, and is only able to extract a speedup of 4 given the intelligent optimized mapper. Note, this graph doesn't preset the raw latency numbers. Even after a $6x$ speedup, the latency of the circuit using unoptimized mapping is significantly worse than the optimized mapper with only a $4x$ speedup.

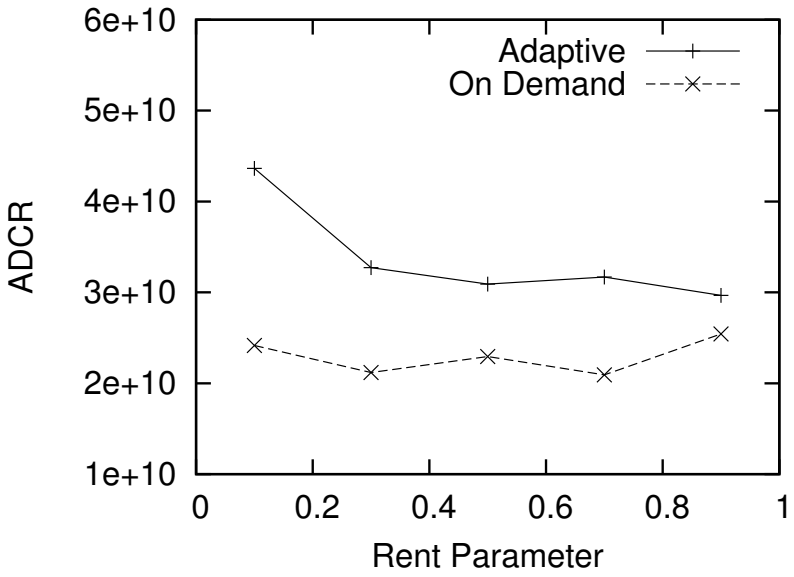



Figure 6.12: Routing performance relative to communication density. Random circuits were generated with varying Rent parameters. Lower rent parameters correspond to a larger computation to communication ratio.

### 6.5.3 Communication Patterns and Circuit Size

The communication pattern within a quantum circuit can vary considerably depending on the structure and size of the circuit. In this section we evaluate how our optimized routing holds up under different communication patterns and circuit sizes. We generate a number of random circuits by varying the number of gates and the Rent parameter described in Section 3.10.2. This process generates circuits with differing amounts and types of communication.

Figure 6.12 shows the difference between Adaptive routing and On-demand routing when varying the Rent parameter. In this case, Adaptive is obtained by using both Minimal Adaptive and Full Adaptive to determine the best schedule. Over all Rent parameter values the Adaptive routing and scheduling improves upon the On-Demand case. However, at lower Rent parameters the Adaptive routing and scheduling improvement is more noticeable. The reason for this difference is that at lower Rent parameters there is more computation that occurs between communication requests. The added time between communication requests provides our Adaptive algorithm more opportunity to use less direct paths to distribute load within the network.

Figure 6.13 shows the difference between the two routing and scheduling techniques given increasing circuit sizes. As the size of the circuits increase, ADCR also increases due to larger area and latency. For all cases, the Adaptive routing is able to obtain an improvement over On Demand by a factor of 1.5 to 2.

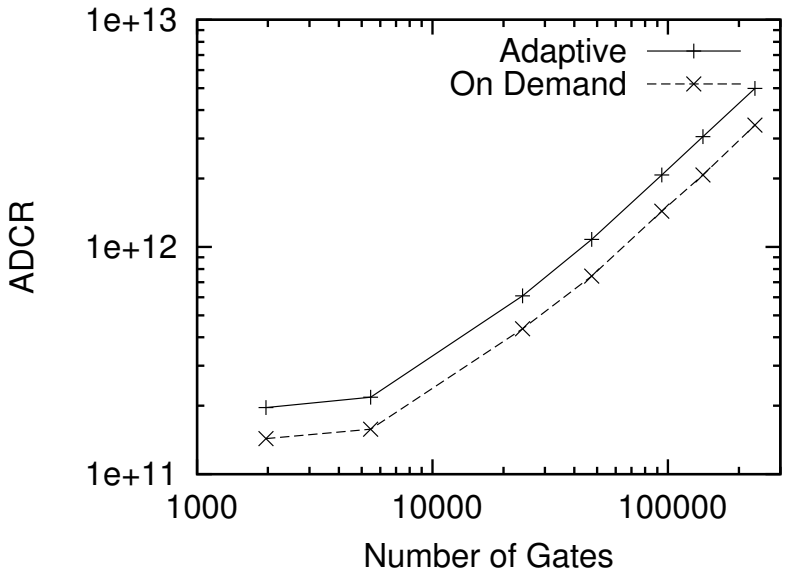



Figure 6.13: ADCR versus the number of gates in a circuit. Random circuits were generated with increasing numbers of gates to determine ADCR for Adaptive and On Demand routing.

## 6.6 Summary

The communication network plays a key role in determining the overall latency of a quantum circuit. We present a number of routing and scheduling techniques to manage communication within the network. Our heuristic prescheduling with minimal and full adaptive routing perform almost as well as optimal, but unlike the optimal algorithm, they are able to scale to large circuit sizes.

From the rest of this work, we run both heuristic prescheduling with minimal adaptive and full adaptive routing to determine the best schedule. We will refer to this best schedule as the *Optimized* case. We will compare all of our results to the On-demand scheduler and refer to it as the *Unoptimized* case.

Our optimized routing and scheduling algorithms take advantage of the data independent portions of connection setup to preschedule connections and reduce total circuit latency. We are able to use a simple greedy heuristic scheduling algorithm to improve on the On-demand case by a factor of 2 to 4 in terms of latency. Additionally, we show that our greedy heuristic generates a schedule and routing that performs within 10% of the optimal case.

# Chapter 7

# Large Quantum Circuits

In previous chapters we developed tools to create low-level ion trap datapaths, schedule short-distance communication, developed an architecture to perform long-distance communication and presented scheduling and routing algorithms to optimize the performance of the communication network. Now we will put all these tools together to analyze the performance of large quantum circuits. Figure 7.1 shows our target application: Shor's factoring algorithm.

Shor's algorithm can be decomposed into two major components: modular exponentiation and a quantum fourier transform (QFT). The bulk of the computation occurs within the modular exponentiation task. In our implementation, modular exponentiation is performed via a series of repeated additions. Therefore, to start our study of large circuits we will focus on the main kernel used to perform Shor's algorithm: a quantum adder. Once we decide on a good adder design, we will use it to build the full Shor's factorization application. The goals of this chapter are to find the best adder circuit implementations, good ADCR-efficient datapaths, and demonstrate how our routing techniques can improve overall circuit performance.

Throughout this section we will compare our *Optimized* routing and scheduling to *Unoptimized* routing (On Demand scheduling). The Optimized value is obtained by using

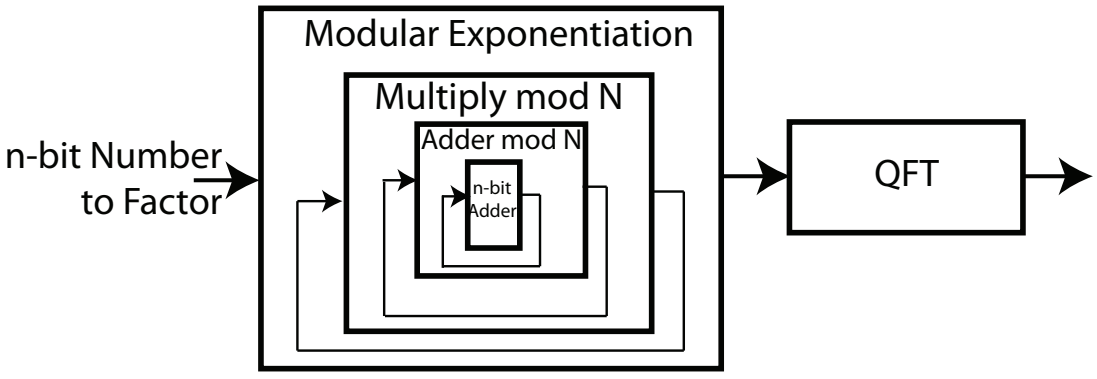



Figure 7.1: *Shor's factoring* architecture.

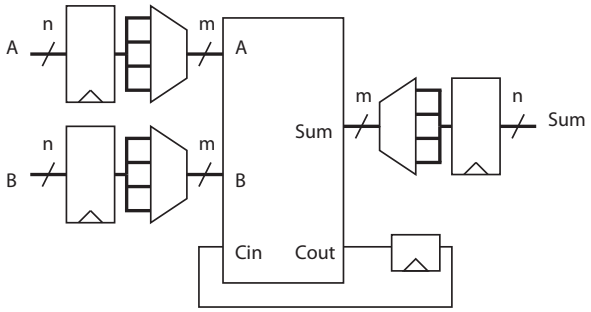



Figure 7.2: Quantum ripple carry adder with "subadder" serialization

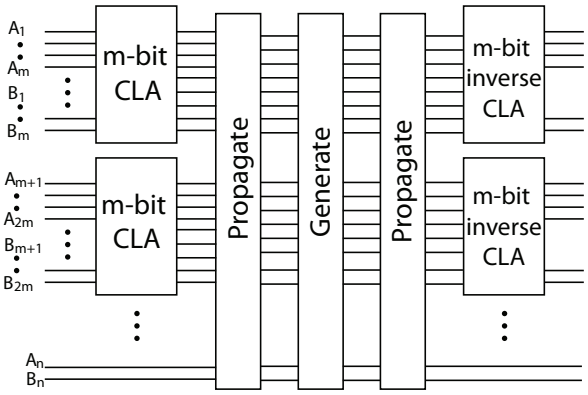



Figure 7.3: Quantum carry-lookahead adder

the best of the two schedules obtained from running heuristic prescheduling with minimal adaptive routing, and heuristic prescheduling with full adaptive routing.

## 7.1 Quantum Addition Circuits

The quantum adder is a fundamental component of Shor's factorization. In this section we will analyze the communication needs of adder circuits and demonstrate how our optimized routing algorithms can improve overall performance. Since we are targeting 1024-bit factorization, we will examine 1024-bit adders.

For our adder implementation we consider the quantum ripple-carry adder (QRCA) [20] and the quantum carry look-ahead adder (QCLA) [21], shown in Figure 7.2 and 7.3 respectively.

A summary of our search for an optimal adder design and datapath is shown in Figure 7.4. Here we see that we are able to obtain the best ADCR using the QCLA circuit with optimized routing. Notably, in our search for an optimal datapath using a QRCA circuit with unoptimized routing, we were unable to locate a single datapath with a success probability greater than zero. The serial nature of a ripple carry adder combined with the delays introduced by the communication network resulted in too much error for the system to tolerate.

### 7.1.1 Ripple-carry Adder

In Figure 7.5 we present ADCR performance for the optimal QRCA optimized datapath. This datapath is the one that generates the optimal ADCR shown in Figure 7.4. Here we can see how the datapath performs as additional network capacity is added. Initially adding network links to the system is beneficial and results in lower ADCR. However, after exceeding a link capacity of 8, the calculated ADCR continues to rise as more link capacity is added to the network. The serial nature of the ripple carry adder essentially

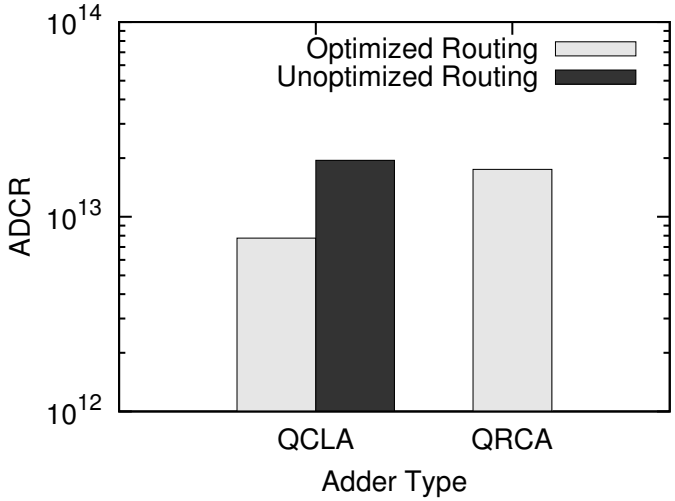



Figure 7.4: Optimal ADCR for 1024-bit QCLA and QRCA. For each adder type and router a number of datapaths were searched to obtain the best ADCR value. All Unoptimized routing QRCA datapaths failed.

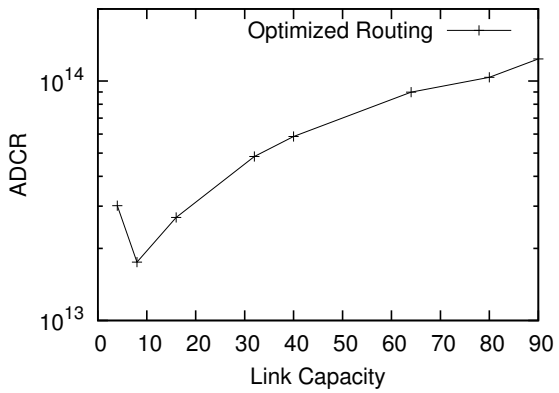



Figure 7.5: QRCA ADCR using the best datapath. Unoptimized routing results in a success probability of 0 and therefore isn't plotted. This datapath results in the best Optimized Routing ADCR corresponding to the bar in Figure 7.4

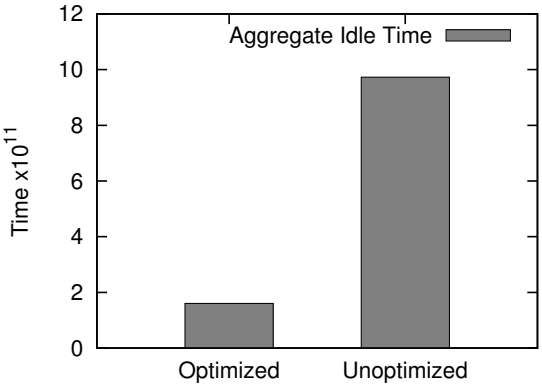



Figure 7.6: Ripple-carry Adder aggregate qubit idle time. The unoptimized router fails due to the large amount of qubit idling in the system.

prevents the routing algorithm from exploiting larger link capacities. As the links are added, total area increases while latency remains constant, resulting in increasing ADCR.

As we noted earlier, all the datapaths searched using unoptimized routing resulted in a success probability of zero. This result is mainly due to the amount of idling data qubits do during the computation as shown in Figure 7.6. The unoptimized routing case results in 6× the amount of time qubits spend idling. The additional idling introduces too much error within the circuit, causing it to fail.

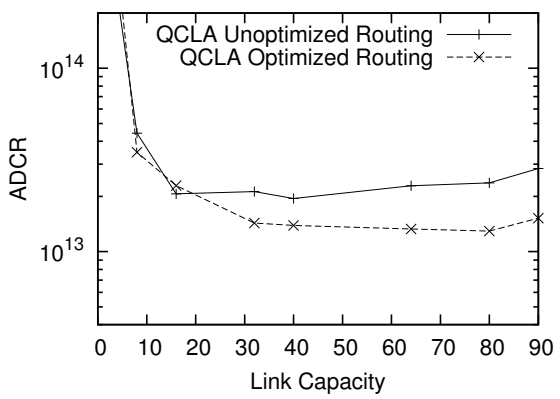



Figure 7.7: QCLA ADCR using the best unoptimized routing datapath. This datapath corresponds to the best unoptimized routing ADCR bar in Figure 7.4.

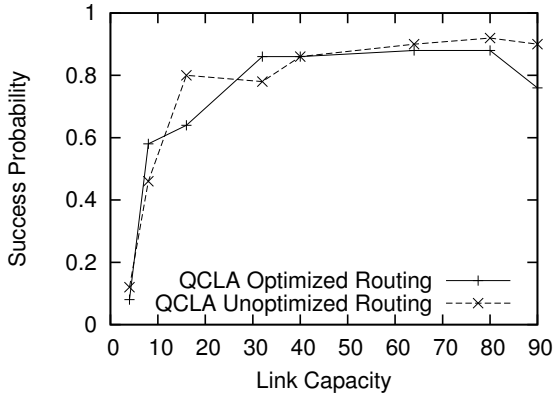



Figure 7.8: QCLA Success Probability vs Link Capacity for the best unoptimized routing datapath. This datapath corresponds to the unoptimized routing ADCR bar in Figure 7.4.

### 7.1.2 Carry Look-ahead Adder

Figure 7.7 shows the ADCR performance for the best QCLA datapath using unoptimized routing. This datapath corresponds to the QCLA Unoptimized routing bar in Figure 7.4. As a comparison, we also plot the ADCR on this datapath if optimized routing is used.

The success probability of this datapath given both optimized and unoptimized routing is shown in Figure 7.8. In this datapath, optimized routing has a lower success probability when compared to unoptimized routing.

Figure 7.9 shows the ADCR performance for the best QCLA datapath using optimized routing. This datapath corresponds to the QCLA optimized routing bar in Figure 7.4. As a comparison, we also plot the ADCR on this datapath if unoptimized routing is used.

The success probability of this datapath given both optimized and unoptimized routing is shown in Figure 7.10. In this datapath, optimized routing has a lower success probability when compared to unoptimized routing.

## 7.2 Shor's Factorization Algorithm

Given what we've learned by analyzing the adder designs, we can now move on to evaluating the full Shor's factorization quantum circuit.

### 7.2.1 Implementation of Shor's

Figure 7.11 shows a block-diagram of our target circuit. It consists of two main components: modular exponentiation and the quantum Fourier transform (QFT). For the modular exponentiation circuit, we rely on the work done in [67] and for the QFT,

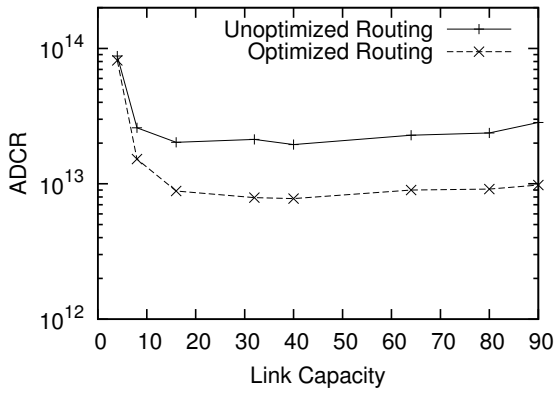



Figure 7.9: QCLA ADCR using the best optimized routing datapath. This datapath corresponds to the Optimized Routing ADCR bar in Figure 7.4.

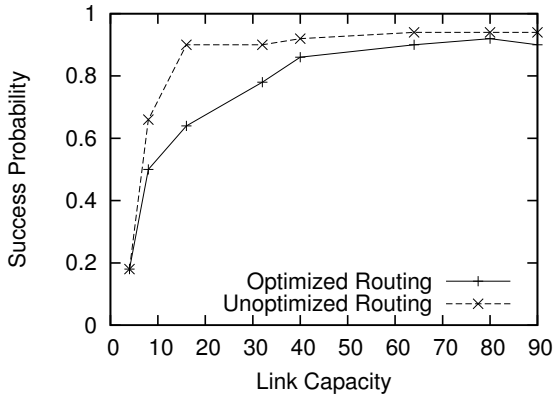



Figure 7.10: QCLA Success Probability vs Link Capacity for the best optimized routing datapath. This datapath corresponds to the Optimized Routing ADCR bar in Figure 7.4.

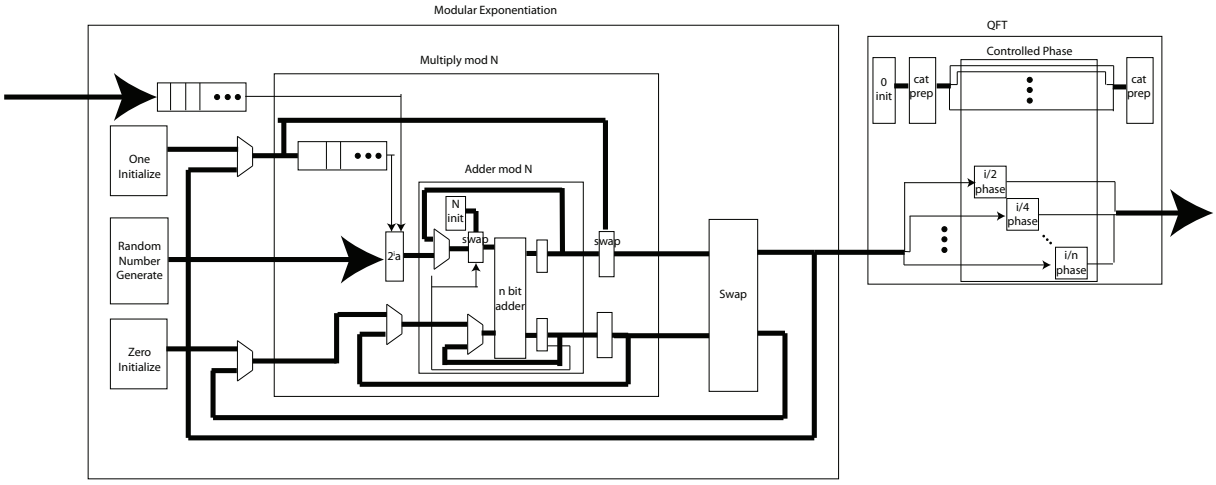



Figure 7.11: *Shor's factoring* architecture. The process consists of two major stages: modular exponentiation and Quantum Fourier Transform. A majority of the complexity is within the modular exponentiation block where the major sub-component is the quantum adder.

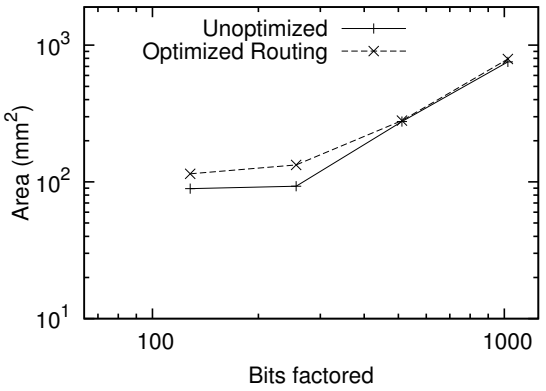



Figure 7.12: Total Area used by Shor's factoring algorithm as a function of the number of bits factored.

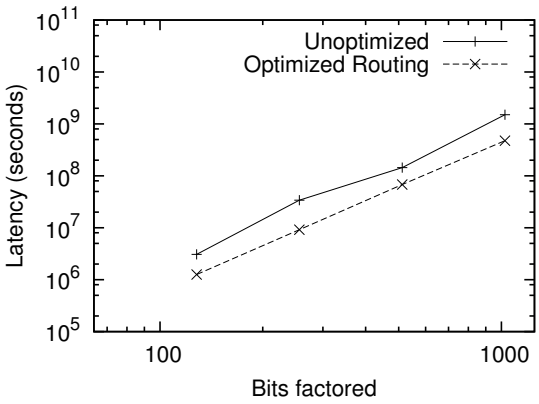



Figure 7.13: Total latency used by Shor's factoring algorithm as a function of the number of bits factored.

[24]. Since addition is a key component of the modular exponentiation circuit, we use our best adder designs from Section 7.1.

### 7.2.2 Performance of Shor's Algorithm

We evaluated circuits for Shor's algorithm based on varying input sizes (128-bit to 1024-bit). For each circuit, we evaluated a number of Qalypso based datapaths to locate the best datapath for the circuit in question and compare our optimized routing algorithms to the unoptimized case. Unfortunately, due to the size of of the Shor's circuits, we were unable to run full error simulations on them to obtain ADCR values. Instead, we optimistically assume the circuits will have a success probability of 1 and simply use $Area \times Delay$ to obtain the optimal datapath.

In Figure 7.12 we see that our optimized routing algorithms require datapaths with slightly larger areas than the unoptimized cases. Since the optimized version has a lower overall latency the operations that occur are spaced much closer together. Consequently, the optimized case requires larger ancilla factories to supply the same amount of ancilla in less time resulting in the increased area numbers.

In Figure 7.13 we see that our optimized routing algorithms improve the latency of the circuits by a factor of 2.4 to 3.6. This improvement is consistent with our adder design results presented earlier. Ultimately, constructing a datapath for a 1024-bit version of Shor's algorithm consumes an area of $880mm^2$ and has a latency of $4.8 \times 10^8$ seconds.

## 7.3 Future Work

This work has investigated communication and control issues in far more detail than prior work and has permitted us to perform a more thorough evaluation of larger circuits than was possible before. While we have addressed details and issues at all levels of

communication and control, there are still a number of area that could benefit from further investigation.

### 7.3.1 Alternate Technologies

Our study has concentrated on ion trap technology as this technology is currently one of the most promising in terms of scalability. A major characteristic of ion trap technology is that movement error is proportional to distance moved. This characteristic may not be the case in alternate technologies. For example, in electron-spins on helium quantum technology [39, 13] movement error is thought to be dramatically less than gate error and less dependent on the distance moved. For these types of technologies, we have to reevaluate the long-distance communication network and use of teleportation.

Additionally, current ion trap technology only operates in two dimensions. It is possible that advances in ion trap technology, or alternate quantum technologies will permit stacking, allowing us to use the third dimension to assist in routing communication channels. Adding an additional dimension warrants studying alternate network topologies. Network structures such as higher dimension hypercubes and butterfly networks may reduce communication set up time and overall circuit latency.

### 7.3.2 Early Network Connections

Our long-distance network is designed to establish a connection for use such that the connection is ready to teleport data as soon as the data completes the prior operation. The connection reserves resources and locks these resources until the data teleports, continuously creating EPR pairs at the source and destination. The reason we keep the connection open the whole time is because we do not want the final EPR pairs used to teleport data to sit idling as it would introduce error. Instead the EPR bits are refreshed with new ones until the data teleport occurs. An alternative would be to consider the case where a connection is setup and closed before the data actually teleports. The final EPR bits used to teleport may decohere slightly as they idle at the end points waiting for data, but we would benefit from releasing the resources in the network. It is worth studying the impact of this idle time to determine if releasing resources early would benefit the overall performance of the circuit.

### 7.3.3 Ballistic Move with Periodic Error Correction

We have determined that relying on low-level ballistic movement to move long distances is impractical in ion trap technology as too much error will accumulate on the data qubits. One technique worth investigating is performing error correction in the middle of longer ballistic move operations. Rather than moving a logical qubit the full distance, we could theoretically move a shorter distance, stop and error correct, and continue moving (possibly stopping multiple times to error correct). This approach will increase the total latency of the move, but may allow us to reduce the size of the long-distance communication network. Unfortunately, performing error correction part way through a move would

require ancilla generation resources in the middle of the communication channels. Adding these resources may eclipse any savings obtained by reducing the size of the long-distance network.

## 7.4 Conclusion

In this work we have presented the first detailed study of communication and control issues within large quantum datapaths. We find that communication plays a significant role in the operation of a large quantum circuit and a functional quantum datapath must contain a robust system to enable reliable communication. To address this issue, we present a scalable architecture for a quantum computer which specifically addresses communication concerns. Our design minimizes communication error by using a specialized teleportation based interconnection network to perform long-distance movement.

We developed a set of tools to construct and study quantum datapath designs based on ion trap quantum technology. Our tools automatically synthesize and insert the interconnection network used for long-distance communication into the target datapath. We carefully design the various components that compose the communication network and present low-level datapaths and control for these components. To optimize the performance of our network, we present a set of greedy heuristics to performance routing and scheduling of communication within the network and show that our approach performs as well as an optimal case determined using integer linear programming.

We study a number of different quantum circuits including randomly generated circuits, quantum adder circuits, and ultimately Shor's factorization algorithm and show that designs using our optimizations significantly improve upon prior work in terms of a probabilistic area delay metric.

# Bibliography


[1] Vikram S. Adve and Mary K. Vernon. Performance analysis of mesh interconnection networks with deterministic routing. *IEEE Trans. on Parallel and Dist. Systems*, 5(3):225–246, 1994.

[2] J.M. Amini, J. Britton, D. Leibfried, and D.J. Wineland. Microfabricated Chip Traps for Ions. *Arxiv preprint arXiv:0812.3907*, 2008.

[3] S. Balensiefer, L. Kregor-Stickles, and M. Oskin. An evaluation framework and instruction set architecture for ion-trap based quantum micro-architectures. In *Proceedings of the 32nd annual International Symposium on Computer Architecture*, pages 186–196. IEEE Computer Society Washington, DC, USA, 2005.

[4] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, 1995.

[5] M. D. Barrett, J. Chiaverini, T. Schaetz, J. Britton, W.M. Itano, J.D. Jost, E. Knill, C. Langer, D. Leibfried, and R. Ozeri. Deterministic quantum teleportation of atomic qubits. *Nature*, 429:737–739, 2004.

[6] C.H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W.K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 70(13):1895–1899, 1993.

[7] C.H. Bennett, G. Brassard, S. Popescu, B. Schumacher, J.A. Smolin, and W.K. Wootters. Purification of Noisy Entanglement and Faithful Teleportation via Noisy Channels. *Physical Review Letters*, 76(5):722–725, 1996.

[8] D.J. Bishop, C.R. Giles, and G.P. Austin. The Lucent LambdaRouter: MEMS technology of the future here today. *Communications Magazine, IEEE*, 40(3):75–79, 2002.

[9] R.B. Blakestad, C. Ospelkaus, A.P. VanDevender, J.M. Amini, J. Britton, D. Leibfried, and D.J. Wineland. High-fidelity transport of trapped-ion qubits through an X-junction trap array. *Physical review letters*, 102(15):153002, 2009.

[10] P.O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan. On universal and fault-tolerant quantum computing: a novel basis and a new constructive proof of universality for Shor's basis. *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 486–494, 1999.

[11] Kenneth R. Brown, Robert J. Clark, Jaroslaw Labaziewicz, Philip Richerme, David R. Leibrandt, and Isaac L. Chuang. Loading and characterization of a printed-circuit-board atomic ion trap. *Physical Review A (Atomic, Molecular, and Optical Physics)*, 75(1):015401, 2007.

[12] CERN. Colt technical computing library in java.

[13] E. Chi, S.A. Lyon, and M. Martonosi. Tailoring quantum architectures to implementation style: a quantum computer for mobile and persistent qubits. In *Proceedings of the 34th annual international symposium on Computer architecture*, page 209. ACM, 2007.

[14] L. Childress and et. al. Fault-tolerant quantum repeaters with minimal physical resources, and implementations based on single photon emitters. *quant-ph/0502112*, 2005.

[15] J. I. Cirac and P. Zoller. Quantum computations with cold trapped ions. *Phys. Rev. Lett.*, 74(20):4091–4094, May 1995.

[16] A. W. Cross and K. M. Svore. A QASM Toolsuite. *http://web.mit.edu/awcross/www/qasm-tools/*, 2006.

[17] A.W. Cross, D.P. DiVincenzo, and B.M. Terhal. A comparative code study for quantum fault-tolerance. *Arxiv preprint arXiv:0711.1556*, 2007.

[18] William J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Trans. on Computers*, 39(6):775–785, 1990.

[19] D. Deutsch, A. Ekert, R. Jozsa, C. Macchiavello, S. Popescu, and A. Sanpera. Quantum Privacy Amplification and the Security of Quantum Cryptography over Noisy Channels. *Physical Review Letters*, 77(13):2818–2821, 1996.

[20] T.G. Draper. Addition on a Quantum Computer. *Arxiv preprint quant-ph/0008033*, 2000.

[21] T.G. Draper, S.A. Kutin, E.M. Rains, and K.M. Svore. A logarithmic-depth quantum carry-lookahead adder. *Arxiv preprint quant-ph/0406142*, 2004.

[22] W. Dür, H.J. Briegel, JI Cirac, and P. Zoller. Quantum repeaters based on entanglement purification. *Physical Review A*, 59(1):169–181, 1999.

[23] M.J. Madsen et al. Planar ion trap geometry for microfabrication. *Applied Phys. B: Lasers and Optics*, 78:639 – 651, 2004.

[24] A.G. Fowler and L.C.L. Hollenberg. Scalability of Shors algorithm with a limited set of rotation gates. *Physical Review A*, 70(3):32329, 2004.

[25] Neil A. Gershenfeld and Issac L. Chuang. Bulk Spin-Resonance Quantum Computation. *Science*, 275(5298):350–356, 1997.

[26] GNU. GLPK (GNU Linear Programming Kit). *http://www.gnu.org/software/glpk/*, 2008.

[27] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.R. Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5(2):287–326, 1979.

[28] S. Guide, M. Riebe, G.P.T. Lancaster, C. Becher, J. Eschner, H. Haeffner, F. Schmidt-Kaler, I.L. Chuang, and R. Blatt. Implementation of the Deutsch-Jozsa algorithm on an ion-trap quantum computer. *Nature*, 421(6918):48–50, 2003.

[29] R.N. Hall, G.E. Fenner, J.D. Kingsley, T.J. Soltys, and R.O. Carlson. Coherent Light Emission From GaAs Junctions. *Phys. Rev. Lett.*, 9(9):366–368, 1962.

[30] W.K. Hensinger, S. Olmschenk, D. Stick, D. Hucul, M. Yeo, M. Acton, L. Deslauriers, C. Monroe, and J. Rabchuk. T-junction ion trap array for two-dimensional ion shuttling, storage, and manipulation. *Applied Physics Letters*, 88:034101, 2006.

[31] D. Hucul, M. Yeo, WK Hensinger, J. Rabchuk, S. Olmschenk, and C. Monroe. On the Transport of Atomic Ions in Linear and Multidimensional Ion Trap Arrays. *Arxiv preprint quant-ph/0702175*, 2007.

[32] David Alexander Hucul. Operation of a two- dimensional ion trap array for scalable quantum computation. B.S. Honor's thesis, University of Michigan, 2006.

[33] N. Isailovic, Y. Patel, M. Whitney, and J. Kubiatowicz. Interconnection Networks for Scalable Quantum Computers. *Computer Architecture, 2006. ISCA'06. 33rd International Symposium on*, pages 366–377, 2006.

[34] N. Isailovic, M. Whitney, Y. Patel, and J. Kubiatowicz. Running a Quantum Circuit at the Speed of Data. *ISCA'08. 35rd International Symposium on*, 2008.

[35] B.E. Kane. A silicon-based nuclear spin quantum computer. *Nature*, 393(6681):133–7, 1998.

[36] E. Knill and R. Laflamme. Theory of quantum error-correcting codes. *Physical Review A*, 55(2):900–911, 1997.

[37] D. Leibfried, B. DeMarco, V. Meyer, D. Lucas, M. Barrett, J. Britton, WM Itano, B. Jelencovic, C. Langer, T. Rosenband, et al. Experimental demonstration of a robust, high-fidelity geometric two ion-qubit phase gate. *Nature*, 422(6930):412–415, 2003.

[38] C.E. Leiserson and J.B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991.

[39] S.A. Lyon. Spin-based quantum computing using electrons on liquid helium. *Physical Review A*, 74(5):52338, 2006.

[40] M.J. Madsen, W.K. Hensinger, D. Stick, J.A. Rabchuk, and C. Monroe. Planar ion trap geometry for microfabrication. *Applied Physics B: Lasers and Optics*, 78:639 – 651, 2004.

[41] Yuriy Makhlin, Gerd Schön, and Alexander Shnirman. Quantum-state engineering with josephson-junction devices. *Rev. Mod. Phys.*, 73(2):357–400, May 2001.

[42] T.S. Metodi, D.D. Thaker, A.W. Cross, F.T. Chong, and I.L. Chuang. A Quantum Logic Array Microarchitecture: Scalable Quantum Data Movement and Computation. *Proc. of Intl. Symp. on Microarchitecture (MICRO)*, 2005.

[43] T.S. Metodi, D.D. Thaker, A.W. Cross, F.T. Chong, and I.L. Chuang. Scheduling physical operations in a quantum information processor. In *Proceedings of SPIE*, volume 6244, pages 210–221, 2006.

[44] C. Monroe, D. M. Meekhof, B. E. King, W. M. Itano, and D. J. Wineland. Demonstration of a fundamental quantum logic gate. *Phys. Rev. Lett.*, 75(25):4714–4717, Dec 1995.

[45] H.C. Nägerl, D. Leibfried, H. Rohde, G. Thalhammer, J. Eschner, F. Schmidt-Kaler, and R. Blatt. Laser addressing of individual ions in a linear ion trap. *Physical Review A*, 60(1):145–148, 1999.

[46] M.A. Nielsen and I.L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2000.

[47] M. Oskin, F.T. Chong, I.L. Chuang, and J. Kubiatowicz. Building quantum wires: the long and the short of it. *Proceedings of the 30th International Symposium on Computer Architecture (ISCA)*, pages 374–385, 2003.

[48] R. Ozeri, C. Langer, JD Jost, B. DeMarco, A. Ben-Kish, B.R. Blakestad, J. Britton, J. Chiaverini, W.M. Itano, D.B. Hume, et al. Hyperfine coherence in the presence of spontaneous photon scattering. *Physical review letters*, 95(3):30403, 2005.

[49] W. Paul. Electromagnetic traps for charged and neutral particles. *Reviews of Modern Physics*, 62(3):531–540, 1990.

[50] C. E. Pearson, D. R. Leibrandt, W. S. Bakr, W. J. Mallard, K. R. Brown, and I. L. Chuang. Experimental investigation of planar ion traps. *Phys. Rev. A*, 73(3), 2006.

[51] J. D. Prestage, G. J. Dick, and L. Maleki. New ion trap for frequency standard applications. *Journal of Applied Physics*, 66(3):1013–1017, 1989.

[52] M. Riebe, H. Haeffner, C.F. Roos, W. Haensel, J. Benhelm, G.P.T. Lancaster, T.W. Koerber, C. Becher, F. Schmidt-Kaler, and D.F.V. James. Deterministic quantum teleportation with atoms. *Nature*, 429(6993):734–737, 2004.

[53] F. Schmidt-Kaler, H. Haeffner, M. Riebe, S. Gulde, G.P.T. Lancaster, T. Deutschle, C. Becher, C.F. Roos, J. Eschner, and R. Blatt. Realization of the Cirac–Zoller controlled-NOT quantum gate. *Nature*, 422(6930):408–411, 2003.

[54] F. Schmidt-Kaler, H. Häffner, S. Gulde, M. Riebe, GPT Lancaster, T. Deuschle, C. Becher, W. Hänsel, J. Eschner, CF Roos, et al. How to realize a universal quantum gate with trapped ions. *Applied Physics B: Lasers and Optics*, 77(8):789–796, 2003.

[55] S. Seidelin, J. Chiaverini, R. Reichle, J.J. Bollinger, D. Leibfried, J. Britton, J.H. Wesenberg, R.B. Blakestad, R.J. Epstein, D.B. Hume, et al. Microfabricated Surface-Electrode Ion Trap for Scalable Quantum Information Processing. *Physical Review Letters*, 96(25):253003, 2006.

[56] P. Shivakumar, M. Kistler, SW Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. *Dependable Systems and Networks, 2002. Proceedings. International Conference on*, pages 389–398, 2002.

[57] P.W. Shor. Algorithms for quantum computation: discrete logarithms andfactoring. *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134, 1994.

[58] P.W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52(4):2493–2496, 1995.

[59] A.M. Steane. Error Correcting Codes in Quantum Theory. *Physical Review Letters*, 77(5):793–797, 1996.

[60] A.M. Steane. Simple quantum error correcting codes. *Phys. Rev. A*, 54:4741–4751, 1996.

[61] A.M. Steane. Space, Time, Parallelism and Noise Requirements for Reliable Quantum Computing. *Fortschritte der Physik*, 46(4-5):443–457, 1999.

[62] A.M. Steane. Overhead and noise threshold of fault-tolerant quantum error correction. *Physical Review A*, 68(4):42322, 2003.

[63] A.M. Steane. How to build a 300 bit, 1 Gop quantum computer. *Arxiv preprint quant-ph/0412165*, 2004.

[64] K. Svore, A. Cross, A. Aho, I. Chuang, and I. Markov. Toward a software architecture for quantum computing design tools. *Proceedings of the 2nd International Workshop on Quantum Programming Languages (QPL)*, pages 145–162, 2004.

[65] K. Svore and et. al. Local fault-tolerant quantum computation. *Phys. Rev. A*, 72:022317, 2005.

[66] D.D. Thaker, T.S. Metodi, A.W. Cross, I.L. Chuang, and F.T. Chong. Quantum Memory Hierarchies: Efficient Designs to Match Available Parallelism in Quantum Computing. *Computer Architecture, 2006. ISCA'06. 33rd International Symposium on*, pages 378–390, 2006.

[67] V. Vedral, A. Barenco, and A. Ekert. Quantum networks for elementary arithmetic operations. *Physical Review A*, 54(1):147–153, 1996.

[68] M.G. Whitney, N. Isailovic, Y. Patel, and J. Kubiatowicz. A fault tolerant, area efficient architecture for Shor's factoring algorithm. *ACM SIGARCH Computer Architecture News*, 37(3):383–394, 2009.

[69] W.K. Wootters and W.H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.

[70] Mark Yeo. The design and implementation of atomic ion shuttling protocols in a multi-dimensional ion trap array. B.S. Honor's thesis, University of Michigan, 2006.