An Investigation into the Realities of a Quantum Datapath

Nemanja Isailovic

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-60
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-60.html

May 11, 2010

An Investigation into the Realities of a Quantum Datapath

by

Nemanja Isailovic

B.S. (University of California, Berkeley) 1999
M.S. (University of California, Berkeley) 2002

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

Spring Semester 2010

The dissertation of Nemanja Isailovic, titled An Investigation into the Realities of a Quantum Datapath, is approved:

Chair	____________________	Date	__________
	____________________	Date	__________
	____________________	Date	__________

University of California, Berkeley

Spring Semester 2010

An Investigation into the Realities of a Quantum Datapath

Abstract

An Investigation into the Realities of a Quantum Datapath

by

Nemanja Isailovic
Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor John David Kubiatowicz, Chair

Quantum computing has shown great potential for being able to solve certain problems which are intractable on classical machines. Peter Shor devised a means to factor large number in polynomial time on a quantum machine, a feat which would compromise modern public key cryptosystems. Further, simulation of quantum mechanical systems, which is exponential in both space and time on a classical machine, is expected to be far faster on a quantum machine. In this work, we present mechanisms for producing a laid out and scheduled quantum datapath tailored to a particular target circuit.

We identify two key pieces of support infrastructure in a quantum datapath. First, some quantum operations require the use of helper qubits known as ancilla qubits which are not part of the target circuit specification. We introduce and design efficient ancilla factories to use as basic functional units in our datapath layouts. Second, we provide designs for the basic components that allow the construction of a teleportation network, which is necessary for long distance communication on a quantum datapath.

We utilize our basic component designs in proposing a malleable architectural specification which we call Qalypso. The benefit of the flexibility of Qalypso lies in the ability to fine tune the various components of the datapath to suit the needs of a given quantum circuit. Ancilla bandwidth, network resources and interfacing of support infrastructure to data may all be tailored to fit circuit characteristics.

To complete the process of laying out and scheduling a quantum circuit, we device heuristics for mapping the circuit onto Qalypso while simultaneously finalizing the datapath characteristics as appropriate for the circuit. Our methods produce a final realizable datapath layout and associated scheduling, both optimized for the circuit in question.

We have implemented these heuristics in a quantum CAD flow toolset currently tailored to designing architectures in ion trap technology. We conclude this thesis by demonstrating the application of these heuristics through the automated toolset to construct a datapath and schedule optimized for Shor's factorization algorithm.

I dedicate this body of work
to my wife, Lisa, for her infinite patience and compassion,
to my mother, father and sister for their support and encouragement
and to my newborn angel Evelyn for the motivation to finish.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I wish to thank my graduate school adviser, John Kubiatowicz, without whose guidance and advice I would not have achieved the success I have found in grad school. I have always been able to depend on Kubi (as he is otherwise known) both for his own endless good ideas and for his encouragement for me to develop my own original but fragmented thoughts into viable designs and results.

I extend immeasurable gratitude to my fellow Kubits, Mark Gregory Whitney and Yatish Patel. The CAD toolset presented in this work has truly been the result of a joint effort with my colleagues, and our continuous discussions and brainstorming sessions have helped me immensely with my own contributions. Further, after so many years of working with them, I now count them among my closest friends and can say that my years in graduate school have been made markedly more fun by their presence in my life.

My thesis committee gets special thanks for making this final stage in my graduate career more fruitful. I thank Professor Kurt Keutzer for good advice concerning classical analogues of my quantum work, and I thank Professor Dorit Hochbaum for helping expand my knowledge of dataflow analysis techniques which have been key to my work.

Much of my early work with quantum circuits was done in tandem with an informal group of graduate students from other universities. I extend my gratitude to Dean Copsey, Tzvetan Metodi and Darshan Thaker from the University of California, Davis for extensive discussions concerning their pioneering work in quantum computer architecture. I thank Andrew Cross and Ken Brown from MIT for helping me get acclimated to various aspects of quantum physics and the ion trap technology early in my graduate career.

Several professors outside of Berkeley have also helped with my work. I thank Fred Chong and Mark Oskin for brainstorming sessions and for their initial work which inspired my interest in quantum computer architecture. I thank the experimental physicists Isaac Chuang and David Wineland for their assistance in formulating realistic assumptions for the ion trap technology which has allowed me to draw more accurate conclusions.

I offer my thanks to Yury Markovsky and Victor Wen for being the “outsiders” who would frequently critique our work to help us target it for wider understanding and applicability. They donated many hours to reading our papers, listening to our talks and providing valuable feedback.

Finally, I would like thank my parents, my sister and my wife for their endless support, love and understanding which has helped me get through the strenuous process of researching and completing this work.

Chapter 1

Introduction

Quantum computers will be able to solve a number of important physics and mathematical problems with interesting asymptotic improvements [67, 2, 54, 29]. The most cited potential use for quantum computing is to factor large numbers based on *Shor's algorithm* [54]. In order for quantum computers to solve these difficult and interesting problems, they will need to support a larger number of quantum data bits (qubits) and quantum circuit elements. In order to sufficiently scale a quantum computer, there are a number of challenges to overcome. We focus on the overall architecture of the quantum datapath, the design of specialized hardware components and the process of mapping a quantum circuit onto a datapath.

To get an idea of the scope of the problem, we begin with a back of the envelope calculation. A typical implementation of Shor's factorization algorithm [56] requires $3n$ data qubits to factor an n bit number. Thus, to factor a 1024-bit number, we need $\sim$3000 qubits. Due to the fragility of quantum state, these qubits must be encoded for redundancy. A popular code is the Steane [[7,1,3]] CSS code, which encodes each data qubit into seven qubits, raising the count to $\sim$21000 qubits. Two pieces of support infrastructure are needed as well. [55] showed that error correction hardware is likely to take an order of magnitude more resources than the qubits being corrected, while [33] showed the same for the communication network. This gives us a final estimate of $21000 + 210000 + 210000 = 441000$ qubits in the system.

In the rest of this chapter, we shall introduce the basics of quantum computing and the state of the art in quantum computer architecture. In Section 1.1, we introduce the fundamentals of quantum circuits, which are intrinsically independent of physical layouts. In Section 1.2, we present the details of the quantum technology we shall be using throughout this work, the ion trap technology.

Since the management of 100's of thousands of qubits will require a great deal of organization, we have constructed a CAD toolset for designing quantum datapaths. In Section 2.1, we introduce the basics of classical CAD tools upon which we shall build. We conclude this chapter in Section 1.3 with an overview of the state of the art in quantum computer architecture, which we shall use as our baseline for comparison.

1.1 Quantum Circuits

We will later talk about how to automatically synthesize, optimize, and lay out quantum circuits. First we must understand what a quantum circuit is and how it differs from the classical circuit model. In order to understand this, we will begin with individual quantum bits (*qubits*) and build up from there.

1.1.1 Qubits, Superposition and Measurement

A quantum bit, or *qubit*, represents the atomic quantity of information in a quantum computer. Unlike a classical bit, which exists in either the 0 or 1 state at any given time, a qubit exists in a mixture, or *superposition*, of the 0 and 1 states. This simultaneous existence in both states may be exploited by carefully devised algorithms.

The standard Dirac notation [44] describes the state of qubit $q0$ in the following form:

$$|q0>= \alpha|0> +\beta|1> \tag{1.1}$$

This means that qubit $q0$ is in a superposition of the states 0 and 1 with the complex coefficients α and β describing the extent to which the qubit is in each state.

It is not possible to directly determine the coefficients in a quantum state. In order to get any useful information out of a quantum system, we perform an operation known as *measurement*, which collapses the qubit to either the $|0>$ or the $|1>$ state. Specifically, when we perform a measurement on the qubit in Equation 1.1, we collapse the state of the qubit to $|0>$ with probability $|\alpha|^2$ or to $|1>$ with probability $|\beta|^2$. Once the measurement is complete, the qubit is in the new collapsed state and the previous coefficients are forever gone. The basis of quantum algorithms is to manipulate the underlying coefficients such that we get a favorable state collapsing upon measurement.

Multi-qubit systems work in a similar manner. Suppose that we have a second qubit $q1$ in the following state:

$$|q1>= \gamma|0> +\delta|1> \tag{1.2}$$

Combining these qubits into a single system is done by taking the product of their states, resulting in the following combined state:

$$\alpha\gamma|00> +\alpha\delta|01> +\beta\gamma|10> +\beta\delta|11> \tag{1.3}$$

The upshot is that the two qubit system has a complex coefficient associated with each possible classical state (in this case, they are 00, 01, 10 and 11). Since the coefficients determine the probability of collapsing to the associated state upon measurement, the sum of the squares of the magnitudes of the coefficients must be equal to one. Also, note that, in general, an n-qubit system requires 2^n complex coefficients to properly describe it, since that is the number of n-bit strings. However, there is no rule against any of the coefficients being equal to zero.

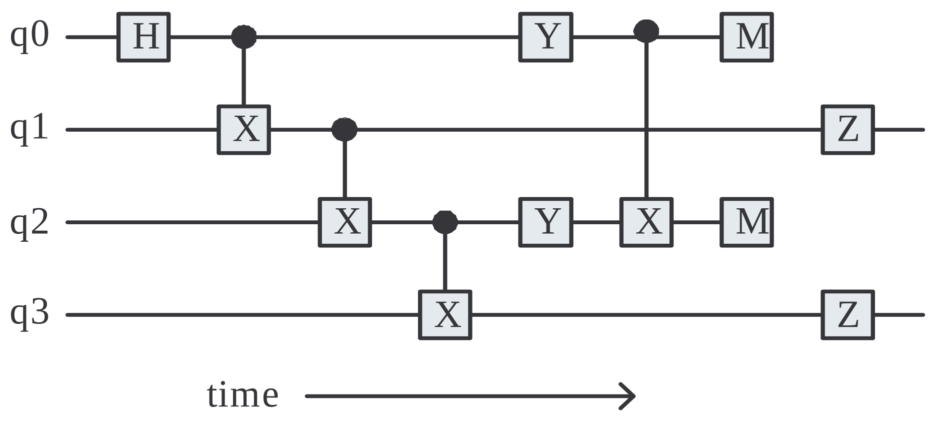


Figure 1.1: An example of a quantum circuit on four qubits with five one-qubit gates, four two-qubit gates and two measurement operations.

1.1.2 Quantum Circuit Model

Classical computers in CMOS technology have spent a lot of time fighting "quantum effects" as feature sizes shrink into the 10s of nanometers range. A quantum computer, on the other hand, strives to amplify and utilize quantum effects such as entanglement and superposition as much as possible. One method for expressing quantum algorithms is via the quantum circuit model. In this model, qubits are represented by horizontal lines, while operations are represented by sequences of quantum gates operating on these qubits.

Figure 1.1 illustrates a quantum circuit on four qubits ($q0$, $q1$, $q2$ and $q3$) with five one-qubit gates (one H, two Y's and two Z's), four two-qubit gates (the ones which connect pairs of qubits) and two measurement operations (the M's). Note that since measurement collapses quantum state to a classical value, the qubit ceases to exist in our circuit and the horizontal line corresponding to that qubit ends at the measurement. The states of the other two qubits, $q1$ and $q3$, still exist at the end of the circuit.

Quantum circuits, which are superficially similar to classical circuits, will be our method for expressing quantum logic in this thesis. The basic abstraction of a quantum circuit is a collection of quantum gates connected by wires. This model is similar to a classical circuit specification but there are two main differences:

- 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 [66] qubits cannot be duplicated. This prevents any fan-out of wires in a quantum circuit.

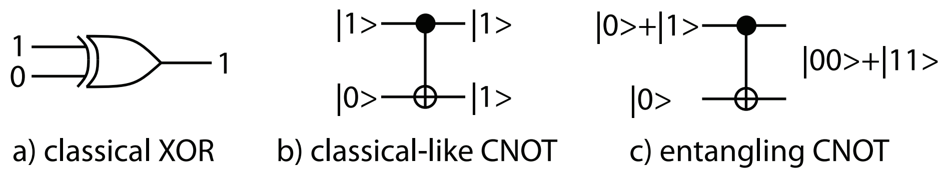


Figure 1.2: A comparison between a classical XOR and its quantum analog: the controlled-not or CNOT. The CNOT gate is reversible, thus the additional output. Figure b) outputs the XOR result to the bottom bit. Figure c) shows the same CNOT when the input is a quantum superposition. In this case the output is an *entangled* qubit state, not representable as independent qubit values for the two outputs.

Figure 1.2a shows quantum and classical XOR gates, the quantum version is known as a controlled-not (CNOT) gate as shown in Figure 1.2b. 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 1.2c, things get more interesting. The resulting output state is an *entangled state* which has no classical analog and is thus unique to the quantum realm.

Quantum circuits operate on these entangled superposition states, and these states provide the key to the power of quantum algorithms. In the end, though, the data cannot stay in a superposition. 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 1.2c. The entangled state $|00>+|11>$ means that when we measure, the resulting classical bit vector will be 00 or 11 (with equal probability).

1.1.3 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 comes from [8] and is 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 1.3 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 is being used.

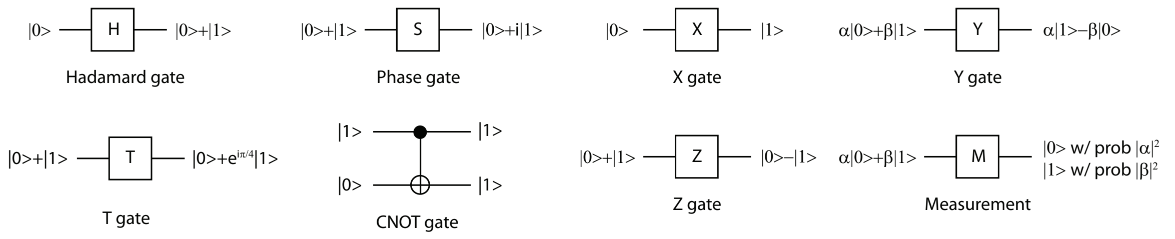


Figure 1.3: Basic gates for quantum circuits: this is a set of gates which supports a universal quantum computing model. The Hadamard gate converts bit values to phase values and vice versa. The phase, T and Z gates rotate the phase of the "1" qubit value by different angles. The CNOT gate is the same as shown in Figure 1.2 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.

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 1.1.6. 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, at the very least in order to be able to read the output of a quantum algorithm.

1.1.4 Encoded Qubits

While the power of quantum superposition enables interesting computing, 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 in classical logic. We must allow a continuum of possible quantum states per qubit instead of two. 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 quantum computing technology in the lab right now 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 op [60]. Compare this to CMOS transistor error rates which range from 10^{-20} to 10^{-15} errors per gate [53].

Typically, quantum errors are abstracted into three different categories:

Gate errors Depending on the physical technology, gates could involve complex sequences of applications of electrical and/or magnetic fields, current, and/or EM radiation applied to one or more co-located qubits. These gate processes

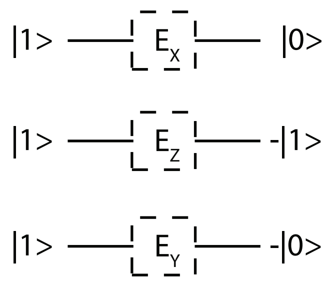


Figure 1.4: Types of errors corrected by quantum error correcting codes: The X error (E_X) flips the bit value, the Z error (E_Z) flips the phase difference between 1 and 0 by π radians, the Y error does both these things, flipping the bit and phase of the qubit.

can introduce errors from apparatus imprecision or tunneling effects between qubits. The abstraction of this error type is that each qubit involved in a gate has some probability of an error being introduced immediately after the gate is finished. Additionally, multi-qubit gates can propagate existing errors from one qubit to another.

Movement/communication errors Qubit communication can involve either physical movement of coherent particles or gate-like operations to transfer state across fixed physical resources. In the former case, kinetic motion of particles can introduce motional heating and even particle loss. The abstraction often used is some amount of distance moved introduces a single qubit error with some probability.

Memory/idle errors Even when a qubit is sitting stationary, interaction with the environment, either through coupling with stray EM fields or contact with stray particles, can cause errors. Since there is no physical action the qubit is performing, memory errors are abstracted to a probability of error per unit of time it is stationary.

Qubit state coherence can quickly decline to unacceptable levels resulting in data loss if this data is unprotected. As in classical computing, redundancy may be used to combat high error rates. This can be done by way of Quantum Error Correction Codes (QECC) [44]. A QECC encodes a data or *logical* qubit, into an encoded block of qubits, similar to a classical error correcting code. It was mentioned before that quantum circuits must preserve a continuum of superposition states, so does this mean that QECC must correct a continuum of errors? Fortunately, the answer is no. These quantum codes are designed such that the error can be measured in the correction process without measuring the data superposition (which would alter the data by collapsing its state). This aspect of quantum code design means that the continuum of errors is collapsed to a choice of 3 different error types (vs. one error

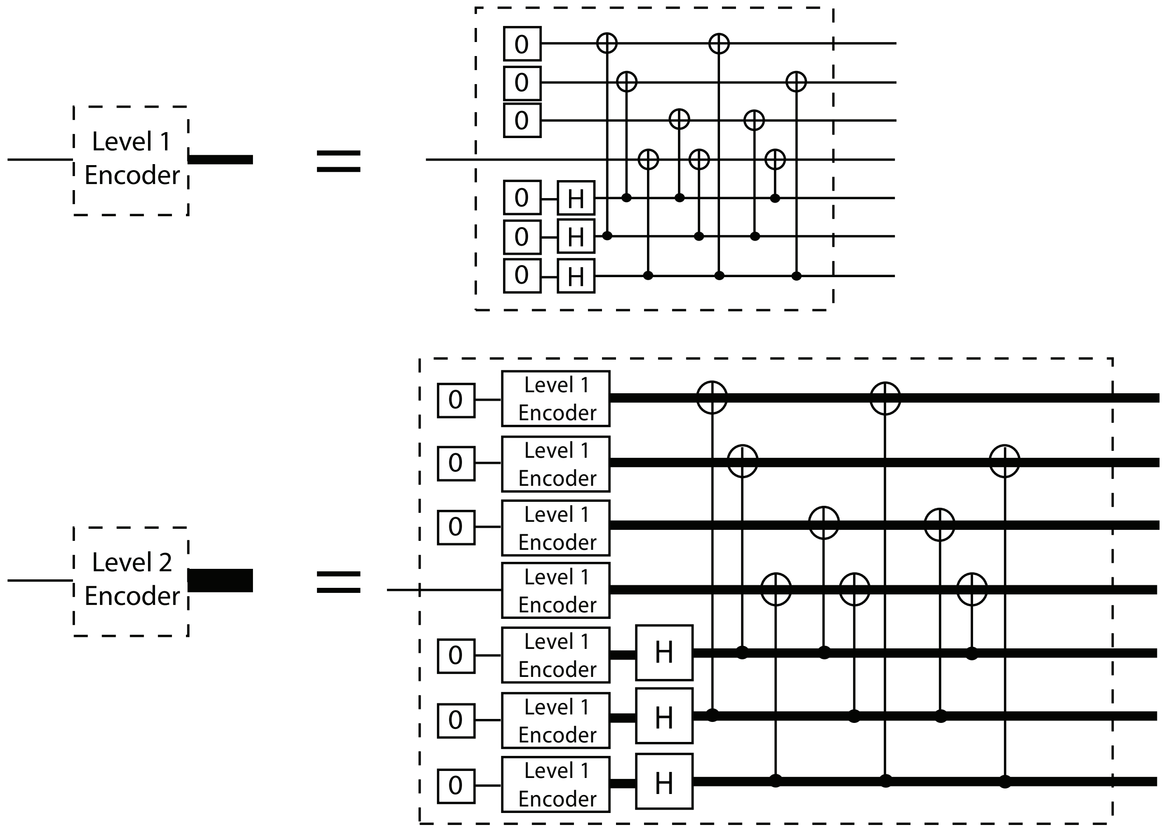


Figure 1.5: On the top, we are encoding a single data qubit into a 7 qubit block code (the [[7,1,3]] CSS code). The boxes with zeros indicate a preparation of a new qubit in the $|0>$ state, if the input qubit is a single physical qubit, this is a level 1 encoder, producing a level 1 logical qubit. The bottom figure is a level 2 encoder, using a level one encoder as a building block to produce a level 1 logical zero valued qubit.

in the classical situation, the bit flip) and the correction process then looks a lot like its classical counterpart. The 3 error types are a bit flip, a phase flip, and both a bit and phase flip as shown in Figure 1.4. The bit flip (E_X) is the same as the classical bit flip error, the phase flip (E_Z)has no classical analogue and is similar to flipping the relative phase between two interfering EM waves.

Encoded blocks can be hierarchically composed so we refer to the level of concatenation as the number of times that a code is recursively applied to the data. Figure 1.5 show a simple recursive encoding scheme using a 7 bit CSS code (explained later in this section). We refer to the lowest order bits in this encoding scheme as *physical qubits*, since these are the bits that correspond to physical two-level quantum systems in the circuit. The "Level 1 Encoder" from the top of Figure 1.5 takes a single physical data qubit and encodes it into 7 physical qubits, making a *level 1 logical qubit*. Assuming that each gate in this encoder has an analogue for operating on a level 1

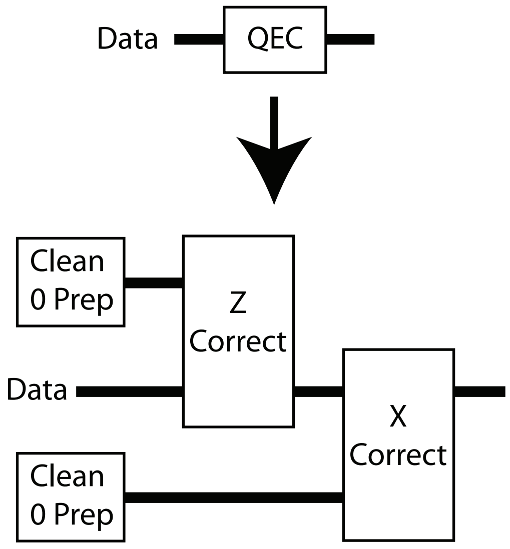


Figure 1.6: Steane-style error correction schemes have the following form: generate two encoded zero states then perform sequential Z and X correction operations.

logical qubit (7 qubit encoded block), we can recursively encode to get a "Level 2 Encoder", shown on the bottom of Figure 1.5. This generates a level 2 logical qubit. The majority of studies into the error correcting abilities of concatenated codes have focused on the asymptotic properties of concatenation of this 7 bit CSS code.

Since many consider gate operations to be the most error prone operations in a quantum circuit [65], decoding a qubit to perform a gate is not feasible. The high error rate of gate operations means that all gates in our quantum circuit must act on encoded data. We refer to these gates acting on encoded data as *encoded gates*. A single encoded or *logical gate* becomes a set of physical gate operations, dependent on which QECC is used.

Even while encoded, quantum data is relatively fragile. In the next two sections, we discuss two key compound operations which are too complex to be performed directly on encoded data with high probability of success: the QEC operation and the operation of any non-transversal gate. In each case, most of the complexity is performed during the creation of encoded ancilla qubits, which are then safely interacted with encoded data to achieve the desired result.

1.1.5 Performing Quantum Error Correction

Using quantum error correcting codes requires that errors are periodically corrected. In classical ECCs, correction involves computing an error *syndrome* based on the data values and then flipping the bit(s) which the syndrome identifies as erroneous. In the QECC case, the correction process is complicated by the fact that

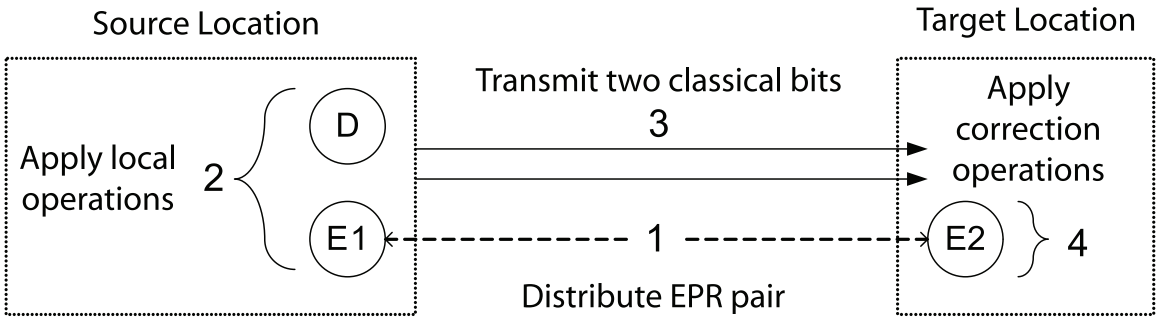


Figure 1.7: Teleporting data qubit D to the target location requires (1) a high-fidelity EPR pair (E1/E2), (2) local operations at the source, (3) transmission of classical bits, and (4) correction operations to recreate D from E2 at the target.

we cannot directly measure the data qubits to obtain their values or we will collapse the superposition and invalidate the computation. Instead, the correction process uses extra *ancilla* qubits that interact with the data qubits, the error information is distilled in these ancilla without transferring any information about the logical data value. Then, the ancilla is measured to get the error syndrome and bit and phase flip corrections (X and Z gates respectively) are applied to the data. Figure 1.6 shows this process.

1.1.6 Teleportation

In order to perform any two-qubit quantum gate, the two qubits must be physically adjacent. Similar to any other quantum operation, qubit movement could be affected by high error rates. Drawing on our work in [35], we make a distinction between short range movement, performed by whatever physical movement primitive is available, and long range movement performed by teleportation.

Teleportation is a useful circuit primitive for long distance communication. Figure 1.7 gives an abstract view of teleportation [6]. We wish to transmit the state of physical data qubit D from the source location to some distant target location without physically moving the data qubit (since that would result in too much decoherence).

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 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 at the destination transform E2 into an exact replica of qubit D's original state [1]. The only non-local operations in teleportation are the

[1]Notice that the no-cloning theorem is not violated since the state of qubit D is destroyed in the

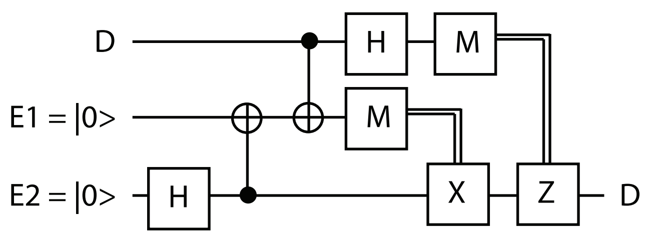


Figure 1.8: Circuit representation for the teleportation operation: The first Hadamard and CNOT gates prepare qubits E1 and E2 in the EPR state. One half of the EPR pair is CNOTed with the data followed by a Hadamard and measurements. The measurement results (classical information represented by double bit lines) are used to apply X and Z gates to adjust the final state.

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). Figure 1.8 shows the circuit representation for this operation (note that E1 and E2 still must be physically separated after the CNOT).

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. Purification, which will be discussed in greater detail in Section 4.1.2, is a process by which multiple low fidelity EPR pairs may be used to generate a single high fidelity pair which may then be used for the teleportation operation. Thus, purification permits a trade-off between channel setup time and fidelity. The trade-off includes extra work to set up the teleportation, but for the benefit of minimal noise on the data qubit being teleported (since most of the work was done on the EPR qubits). Since EPR pair distribution can be performed in advance, qubit communication time can approach the latency of classical communication; of course, channel setup time grows with distance as well as fidelity.

As discussed in our work in [33], qubits were clustered into regions in which we used physical transportation for data qubits and then inter-cluster data movement was done via teleportation. Dedicated teleportation units were used for each cluster and EPR distribution channels that provided the needed ancilla were set up between the units. These EPR channels were structured to use only physical movement or a combination of physical movement and chained teleportation across shorter distances.

1.2 Quantum Computing Technologies

The substrate technology we choose for our study is based on *trapped ions* [12, 42]. In this section, we will discuss the basic operation of an ion trap quantum computer and allude to the various issues that arise when trying to control the system.

process of creating E2's state.

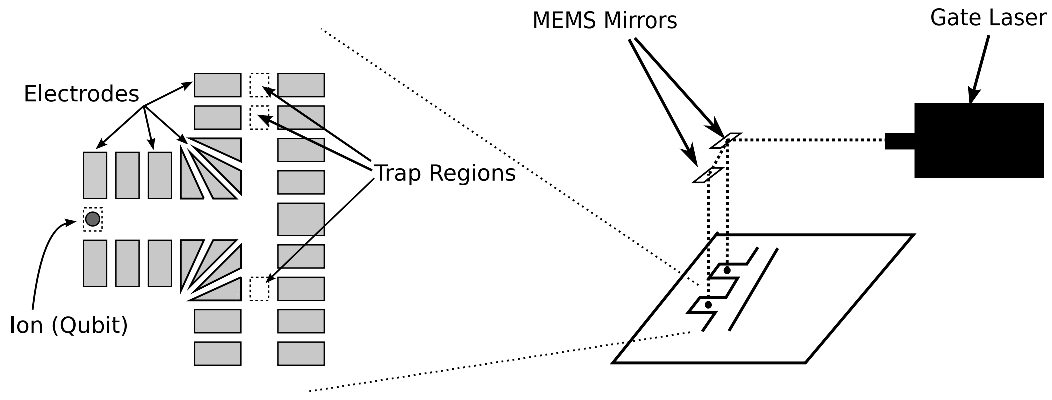


Figure 1.9: Simplified ion trap technology view. Ions (qubits) are trapped between electrodes in the trap regions. Ballistic movement of ions is performed by changing the voltages of the electrodes. A laser is routed to the location of the ions to perform a gate.

We highlight aspects of the system that will require novel architectural decisions to control.

1.2.1 Ion Traps at a Glance

The target technology for our toolset will be ion traps, which has shown potential for scalability [36]. In this technology, a physical qubit is an ion, and a gate is a location wherein an ion is trapped so it may be operated upon. The ion is both trapped and ballistically moved by applying pulse sequences to discrete electrodes which line the edges of ion traps (see the left of Figure 1.9). The ion moves along in a potential well created by the control electrodes.

Gate operations are performed by precise laser pulses aimed at trapped ions. Measurement of a qubit is performed by exciting the target ion with a different frequency laser pulse and then detecting fluorescence using a CCD. Laser beams can be split and routed by an array of MEMS mirrors to simultaneously fire on multiple gate locations, thus allowing SIMD gate operation [37] (Figure 1.9 from [48]). A working ion trap quantum computer will require coordinated control of trap electrodes, lasers, CCDs, and micro-mirrors to perform the proper concurrent operations to implement quantum gates. Here is a break down of these components.

1.2.2 Trap Electrodes

In order to properly confine a qubit (ion) in a trap, the electrodes around the qubit must be precisely controlled to ensure that the ion does not unexpectedly move into adjacent traps, fly out of the top of the trap channel into the vacuum, or adhere to the electrode surfaces. Ballistic movement of qubit ions between traps is likewise achieved by coordinated application of changing voltages to all the electrodes near the ion being moved to generate the precise electrostatic attractive and repulsive forces necessary.

For instance, the process of moving a qubit around a corner involves a procedure which uses sequences of five pulses with at least four discrete voltage levels on about 15 different electrodes [24]. Coordinating a large number of physical ion qubits necessary to do a useful computation (at least in the 100,000s) would require concurrent control of a million electrodes. As we just mentioned, control of each electrode will be more complicated than just turning it on and off, as it will have more than two voltage levels. Due to the need for a million trap electrode controllers to drive the appropriate voltages, these structures can benefit greatly from logic reuse.

Fortunately, there is substantial regularity in both the trap layouts we use to design the computer and the ion qubit movement through these regular structures. We could imagine grouping adjacent sets of traps into blocks, similar to the layout proposed in [3]. Trap blocks with the same geometry could be controlled by the same replicated logic block. The block controller could then accept directives like "turn qubit around corner" or "hold qubit". Adjacent blocks would then have a simple handshake that would enable local decisions on when one trap block is done confining a qubit ion and when its neighbor takes control. In addition to simplifying global control of electrodes, this could also also eliminate potential skew problems in a monolithic controller, which could result in qubit ion errors or even total loss of a qubit.

1.2.3 Gate Lasers

While trap electrodes are all that are needed to move qubit ions, quantum gates are performed by moving qubit ions to designated traps and then applying laser pulses to them. [50] and [51] have detailed listings of the laser pulses used in an experiment to apply a set of one and two qubit gates. 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 precise amount of time. [30] and [51] show the qubit ion energy state transition curves under laser application. 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 [43], two qubit gates require qubit ions to be adjacent in a single ion trap with a distance between them around 7-20μm, leading to a requirement of a beam width of around 5μm. In this particular experiment, obtaining such a resolution was achieved with a rather large Nikon lens. Also mentioned in [43] is the need for a laser with a very stable frequency, one that is within 1-kHz of the transition frequency between qubit ion energy levels. This 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 [7] 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 actual lasers. This 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. This 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.

1.2.4 Measurement

A measurement operation consists of the application of another laser frequency, separate from those needed for gates, and the collection of light fluoresced from the ion with a CCD camera. If an ion is measured in the one state, it fluoresces, if it is in the zero state, it does not. Thus, one must observe whether a particular ion trap location is emitting light when the measurement laser is applied.

[43] shows an apparatus set up to perform this collection via CCD. Following the SIMD model for all our operations on qubits due to laser scarcity, we assume the use of a single large, high-resolution CCD camera positioned above the entire ion trap computer, with a lens between them to resolve the micrometer scale distances between fluorescing ions. All the measurement lasers are applied synchronously, and once enough time has passed to collect sufficient photons, we read out the image on the CCD and process it to determine which sites were fluorescing.

[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 [31].

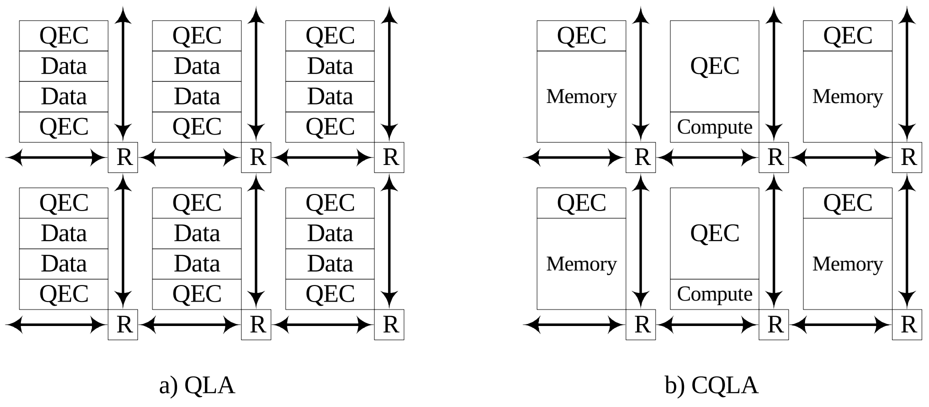


Figure 1.10: Two architectures from prior work: a) The Quantum Logic Array (QLA) consists of a mesh grid of tiles linked by routers (R) to implement long distance communication. Each tile has room for two encoded data qubits and quantum error correction (QEC) resources for each. b) The Compressed Quantum Logic Array diversifies into two types of tiles: Memory tiles in which qubits are idle and don't need to be error corrected as often and Compute tiles which require relatively more QEC resources per data qubit.

With this knowledge about ion traps in hand, we can look at how we would build these physical structures into larger quantum circuits.

1.3 Quantum Computer Architecture

Each qubit is a physical entity in the system, and they must all be managed simultaneously. To accomplish this, we need to make intelligent architectural decisions. Figure 1.10 shows the state of the art in prior work on quantum datapaths, both of which are logical extensions of classical approaches. The Quantum Logic Array (QLA) [23] is essentially an FPGA, with each tile being able to perform any one and two qubit encoded gate, plus the capability to error correct both data qubits. The tiles are connected by a mesh grid of routers.

The Compressed Quantum Logic Array (CQLA) saves area by exploiting the fact that circuit parallelism may be limited, and thus not every data qubit needs to be operated upon simultaneously. CQLA has Compute tiles where data operations are performed and Memory tiles where data reside when idle. Since operations are far more error prone than idleness, QEC resources may be significantly reduced in Memory regions.

Both QLA and CQLA suffer from the drawback that very little reasoning was given for these architectural choices other than intuition. Encoded ancillae are needed for error correction, so they are provided uniformly everywhere; a teleport network is

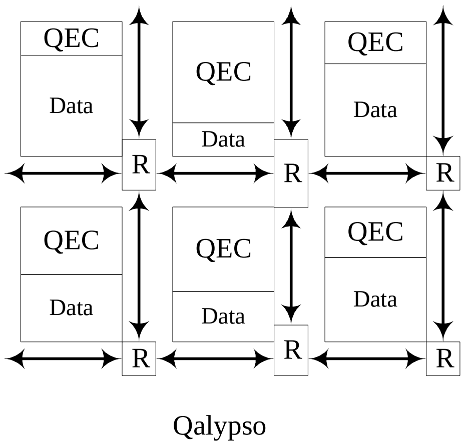


Figure 1.11: Qalypso is a malleable architecture consisting of components similar to those in (C)QLA. By allowing the datapath to be varied somewhat while mapping a quantum circuit, we may better fine tune the end result.

needed, so a basic one is provided, though sizing is never justified; classical machines have memory regions, so the idea is applied to CQLA; etc. Aside from selecting the size of memory relative to compute regions in CQLA, these proposals are not tailored to the circuit in question.

In this work, we propose Qalypso (Figure 1.11), a malleable architecture in which the individual components may be resized automatically via a computer aided design flow to better tailor the end result to the circuit in question. Thus, no Memory regions are preallocated, but different tiles may end up with vastly differing computational capabilities. This allows us to better size the support infrastructure, both ancilla factories and the teleport network, such that the hardware resources are more highly utilized, resulting in less wasted space on the datapath.

1.3.1 Hardware Components

Since quantum programs are specified in terms of low level physical operations, prior work has viewed the quantum datapath as an isotropic sea of physical gate locations, akin to a classical datapath consisting of a “Sea of NANDs.” However, since our objective is to design hardware tailored to a target circuit, we wish to have specialized units in our datapath. To this end, we identify three basic components of a quantum datapath.

First, in order to execute the desired circuit, we need areas for data to reside and to communicate with other data in order to perform multi-qubit operations. In prior work, these areas of the datapath have been referred to interchangeably as data regions and compute regions, and we shall continue to use this terminology.

Second, it is generally believed that QEC operation count in a quantum machine will dwarf the operation count of the target circuit, and it is known that generation of encoded helper qubits known as *ancilla qubits* dominate the process of QEC. Further, the ancilla generation process, which is performed prior to interaction with the data qubit being corrected, is identical across QEC steps. Thus, the ancilla generation subcircuit is duplicated many times during the evaluation of a quantum circuit. For this reason, in Chapter 3 we tackle in detail the layout and scheduling of the ancilla generation circuit, culminating in Section 3.3.2 in our design of and justification for a pipelined ancilla factory, which we then utilize in our Qalypso architecture.

Third, we described in Section 1.1.6 the process of teleportation, which may be used for relatively low latency and error communication over long distances. This teleportation setup process allows for a single point to point communication. Providing for generally available long distance communication channels across the entire datapath, however, necessitates a teleportation based communication network. We discuss in Section 4.1 our designs for the basic components of this network as well as our model for combining these components into a working system.

1.3.2 The Mapping Problem

With support infrastructure components and a malleable architecture in hand, our final goal in this work is to map the quantum circuit onto Qalypso. The QLA approach to mapping is to move the target qubit to the source qubit's location for each two qubit gate. The CQLA approach is to use the limited compute region locations in a greedy fashion with idle qubits being knocked out to memory as necessary. In both cases, the authors assumed ubiquitous QEC ancilla production and a teleportation network capable of handling all long distance communication, which are unrealistic assumptions for these major pieces of support infrastructure.

We focus on three areas of problems/opportunities:

1. The support infrastructure (including both ancilla factories and the teleport network) may be designed at any point along the area-delay trade-off, not necessarily simply sized to meet all deadlines in a mapping. As the support infrastructure is much more resource-intensive than the main computation, it contributes a great deal to our final evaluation of a mapping.

2. The datapath is malleable, which opens up possibilities but also greatly expands the search space.

3. In the quantum realm, a data qubit remains in a gate location after the gate is completed, precluding the use of that gate location by other qubits. The data qubit must be actively moved away in order to "release" the hardware resource.

1.3.3 Metrics

To evaluate the quality of our results, we need metrics of success on which to compare our designs. Typically, we are interested in evaluating the quality of circuit optimization and layout. The final layout of an optimized circuit is what determines the cost and performance of a device. The metrics to consider here are:

Area The overall size of a layout directly impacts ease and cost of fabrication. Also, smaller layouts can contribute to lower power usage and less communication delay due to shorter wires.

Delay In many cases, runtime performance is the most important measure of a design. Getting more work done in less time is often the driving factor for new designs and optimizations.

Reliability Device reliability is becoming more of a concern as transistor feature sizes get closer to atomic scales, since quantum effects as well as fabrication equipment precision become more of an issue. There are a broad array of techniques to improve reliability, at all different levels of granularity in the design. In the end, we are interested in the overall probability that a permanent or transient fault will corrupt data such that incorrect output is obtained from the device.

Each of these metrics may be considered individually. However, we would ideally like a single metric to optimize. Classically, area-delay product is a commonly used metric. We would like to have a similar metric for our quantum designs which incorporates the effect of failure probability on the true latency to a correct result. To this end, we introduce our own aggregate metric here.

Area-Delay-to-Correct-Result As mentioned earlier, delay and success probability are closely connected in the evaluation of any quantum circuit, this is because 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:

$$\begin{aligned} E(Delay) &= Delay_{singlerun} \times E(runs\ for\ correct\ result) && (1.4) \\ &= Delay_{singlerun} \times \sum_{n=1}^{\infty} \frac{n}{P_{success}(1-P_{success})^{n-1}} && (1.5) \\ &= Delay_{single_run} \times \frac{1}{P_{success}} && (1.6) \end{aligned}$$

Prior work has focused on maximizing the overall success probability $P_{success}$ at all costs. This 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; if we can reduce a layout's delay by 10x with only a 10% reduction in success probability, this is probably a good trade-off. 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 to get 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:

$$\begin{aligned}\text{ADCR} &= \text{Area} \times E(\text{Latency}_{\text{total}}) &(1.7)\\ &= \text{Area} \times \sum_{n=1}^{\infty} n \cdot \text{Latency}_{\text{single}} \cdot P_{\text{success}}(1 - P_{\text{success}})^{n-1} &(1.8)\\ &= \text{Area} \times \frac{\text{Latency}_{\text{single}}}{P_{\text{success}}} &(1.9)\end{aligned}$$

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, alternatively, layouts that use the same area for lower latency or higher success probability.

1.4 Thesis Roadmap

In this work, we present heuristics for the automated generation of a layout and associated scheduling optimized for a given target quantum circuit. In order to evaluate our heuristics, we have implemented a CAD flow for quantum circuits. In Chapter 2, we describe our toolset and how it differs from a classical CAD flow. We demonstrate in Chapter 3 how our tools may be used to design custom modules of subcircuits which may then be used hierarchically in larger designs. Specifically, we design ancilla factories which are necessary for both quantum error correction and for the operation of certain quantum gates.

In Chapter 4, we investigate the search space of the physical layout and introduce Qalypso, our microarchitectural model for a quantum datapath. Chapter 5 presents our heuristics for mapping a quantum circuit onto Qalypso and simultaneously finalizing the datapath. Finally, in Chapter 6, we put it all together by applying our tools to practical quantum adder circuits and then to the complete circuit for factorization of a 1024-bit number, which involves approximately one trillion logical gates. We demonstrate a 5x improvement in ADCR over the baseline heuristics.

Chapter 2

Overview of Computer Aided Design for Quantum Circuits

When building a CAD toolset for quantum computer design, we may leverage many lessons learned from classical CAD flows. For this reason, we begin this chapter with a high-level overview of classical CAD. Following that, we summarize the differences between classical and quantum datapath design which must be addressed by a quantum flow. The remainder of the chapter provides many of the details of our CAD flow implementation.

2.1 Classical Computer Aided Design Flows

The studies built upon in this work rely on techniques from computer aided design (CAD) tools for classical circuits [15, 52]. Before we discuss our quantum design flow, we review some of the parts of a classical CAD flow. A vastly simplified version of a typical classical circuit CAD flow is shown in Figure 2.1. The purpose of the flow is to take in some sort of abstract circuit specification and produce a physical design that can then be fabricated. The abstract spec is usually given in languages like Verilog [62], VHDL [41], or higher level languages like SystemC [28]. The physical design depends on the underlying technology but usually consists of some sort of geometric specification of gate-like and wire-like configurations.

2.1.1 Logic Synthesis and Optimization

The logic synthesis step of a CAD flow is to take a high level specification of the circuit's behavior and create a network of basic logic gates and connections between them that faithfully implements the high level specification. The resulting gate network is typically loosely tied to the basic operations available on the technology we are synthesizing for.

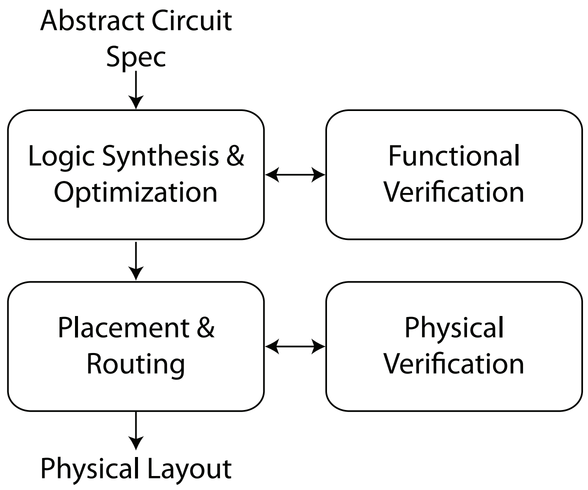


Figure 2.1: Simplified view of a classical computer-aided design flow. A user-specified application circuit specification is first synthesized into some sort of gate network, then physical components are geometrically mapped to a substrate to make physical design. Verification steps ensure equivalence between stages.

Since the high level constructs could potentially have many functionally equivalent translations to a gate network, the synthesis and optimization stage can iterate through many different networks, trying to minimize circuit latency, gate count, etc.

2.1.2 Functional Verification

Due to the potentially complex interactions from different synthesis and optimization methods in the previous step, it is desirable verify that the synthesized gate network works the same way as the user specification. This step can consist of a variety of methods including *formal verification*, where a mathematical models of the two designs are compared or *logic simulation* where the designs are simulated with identical test inputs to make sure they produce the same output.

2.1.3 Placement and Routing

The individual gates in the synthesized network are mapped into physical elements that are geometrically onto a substrate. The goals of this *placement* stage is to convert each gate element into a physical element in the given technology and to place gates that are directly connected in the network in close proximity to each other. After this, wires are *routed* between the physical gates. Since both wires and gates take up physical space, the placement and routing stages are iterative and converge on a design where everything fits onto the substrate.

2.1.4 Physical Verification

Next, this physical design must be verified that it meets requirements imposed by the fabrication technology. This verification of requirements is typically done through *design rule checking* against a detailed set of rules of allowable element geometries. It also must match the functionality of the gate network from the previous stage. This verification of functionality is typically done by extracting a more abstract circuit representation from the physical design and comparing it to the gate network.

2.2 Differences Between Classical and Quantum CAD Flows

To understand how the classical CAD paradigm must be adapted to suit the quantum realm, we must first understand 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, this 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.

Synchronous Gate Operations Related to the previous item, since a classical control network is available, all quantum gates can be essentially synchronous. The ability to classically control these synchronous gates means that we can reuse physical gate locations in a layout to perform more than one gate in the circuit specification. Such gate reuse provides the opportunity to reduce movement (reducing movement error) and reduce the 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 underlying technology used to lay out our circuits. As mentioned in Section 1.2.1, 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 actually 4-way intersections where only one direction can be operational at a time.

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 [48, 24].

The quantum electronic design automation (EDA) system we have developed is modeled after a classical EDA tool flow but accounts for many of these differences between quantum and classical circuits in general and ion trap quantum computing in particular. Figure 2.2 shows the currently available components in our CAD flow.

The highlighted components are the focus of this work and will be covered in great detail. Custom Modules are fully laid out and scheduled designs which may be used as subcircuits of larger circuits. Ancilla generation is a prime example of a commonly occurring subcircuit, and Chapter 3 will cover our process for designing ancilla factories as custom modules which are then used in the rest of this work.

Microarchitectural variations are the subject of Chapter 4, wherein we investigate the search space for the layout of the datapath of an ion trap quantum machine. The problem of mapping the operations of a quantum circuit onto a fixed datapath is somewhat analogous to the problem of Placement in the classical realm. This mapping problem is the subject of Chapter **??**.

The non-highlighted components are required for a complete flow but are not the focus of this work. They are described in the remainder of this chapter.

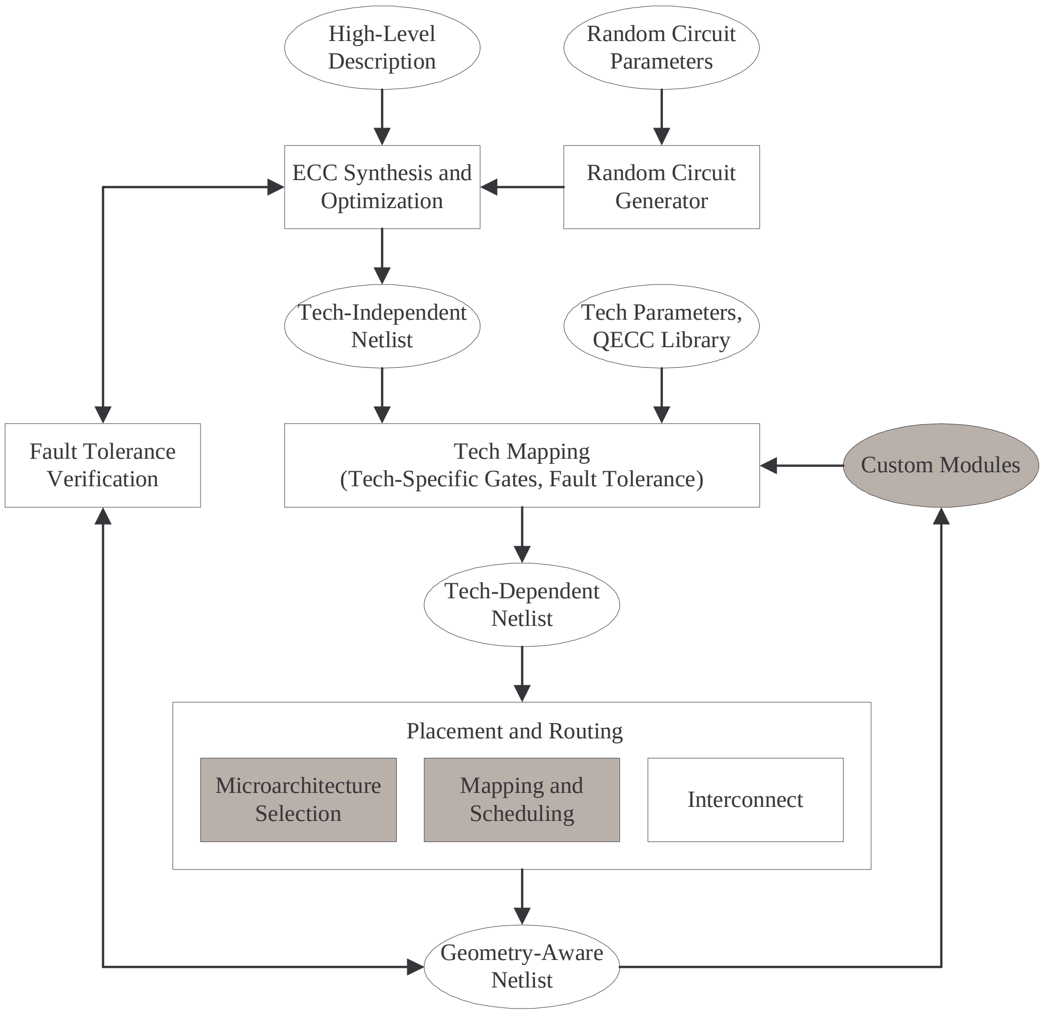

High-Level Description
Random Circuit Parameters
ECC Synthesis and Optimization
Random Circuit Generator
Tech-Independent Netlist
Tech Parameters, QECC Library
Fault Tolerance Verification
Tech Mapping (Tech-Specific Gates, Fault Tolerance)
Custom Modules
Tech-Dependent Netlist
Placement and Routing
Microarchitecture Selection
Mapping and Scheduling
Interconnect
Geometry-Aware Netlist

Figure 2.2: A high level view of our computer-aided design flow for quantum circuits. The highlighted blocks denote the contributions focused on in this work.

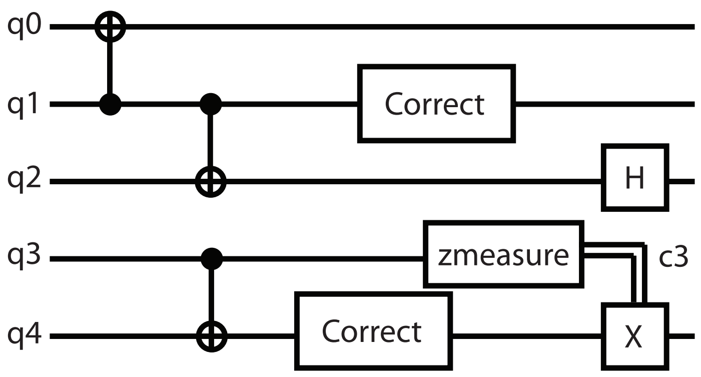
2.3 Application Circuit Specification

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 in [3]. QASM is similar to the classical MIPS assembly language [32]. Basic quantum operations and qubit operands, similar to classical registers, are listed in order in which they are supposed to be executed.

The full QASM instruction set that we use is shown in Table 2.1. 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 `correct` operations are not error prone because they

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 2.1: Summary of all the quantum instructions we use.



```
cx q1, q0;
cx q1, q2;
correct q1;
h q2;
cx q3, q4;
zmeasure q3, c3;
correct q4;
(@c3==1) x q4;
```

Figure 2.3: A quantum circuit and the equivalent QASM instruction stream representing it.

only virtual instructions to do actual error state updates and do not correspond to the entire error correction process which contains many error-prone physical gates.

Figure 2.3 shows an example of a quantum circuit and its QASM specification. In this 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:

```
qubit cin;
qubit cout;
qubit a;
qubit b;

input cin;
input a;
input b;
toffoli cout, a, b;
cx b, a;
toffoli cout, cin, b;
cx b, cin;
output a;
output b;
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:

```
module carry cin, a, b, cout {
  toffoli cout, a, b;
  cx b, a;
  toffoli cout, cin, b;
};

module carry_inv cin, a, b, cout {
  toffoli cout, cin, b;
  cx b, a;
  toffoli cout, a, b;
};

module sum cin, a, b {
  cx b, a;
  cx b, cin;
};

qubit cin0, cin1, cin2, cin3, cout;
qubit a0, a1, a2, a3;
qubit b0, b1, b2, b3, b4;

carry cin0, a0, b0, cin1;
carry cin1, a1, b1, cin2;
carry cin2, a2, b2, cin3;
carry cin3, a3, b3, b4;
cx b3, a3;
sum cin3, a3, b3;
carry cin2, a2, b2, cin3;
sum cin2, a2, b2;
carry cin1, a1, b1, cin2;
sum cin1, a1, b1;
carry cin0, a0, b0, cin1;
sum cin0, a0, b0;
```

Note that when calling modules, the register names must effectively be renamed

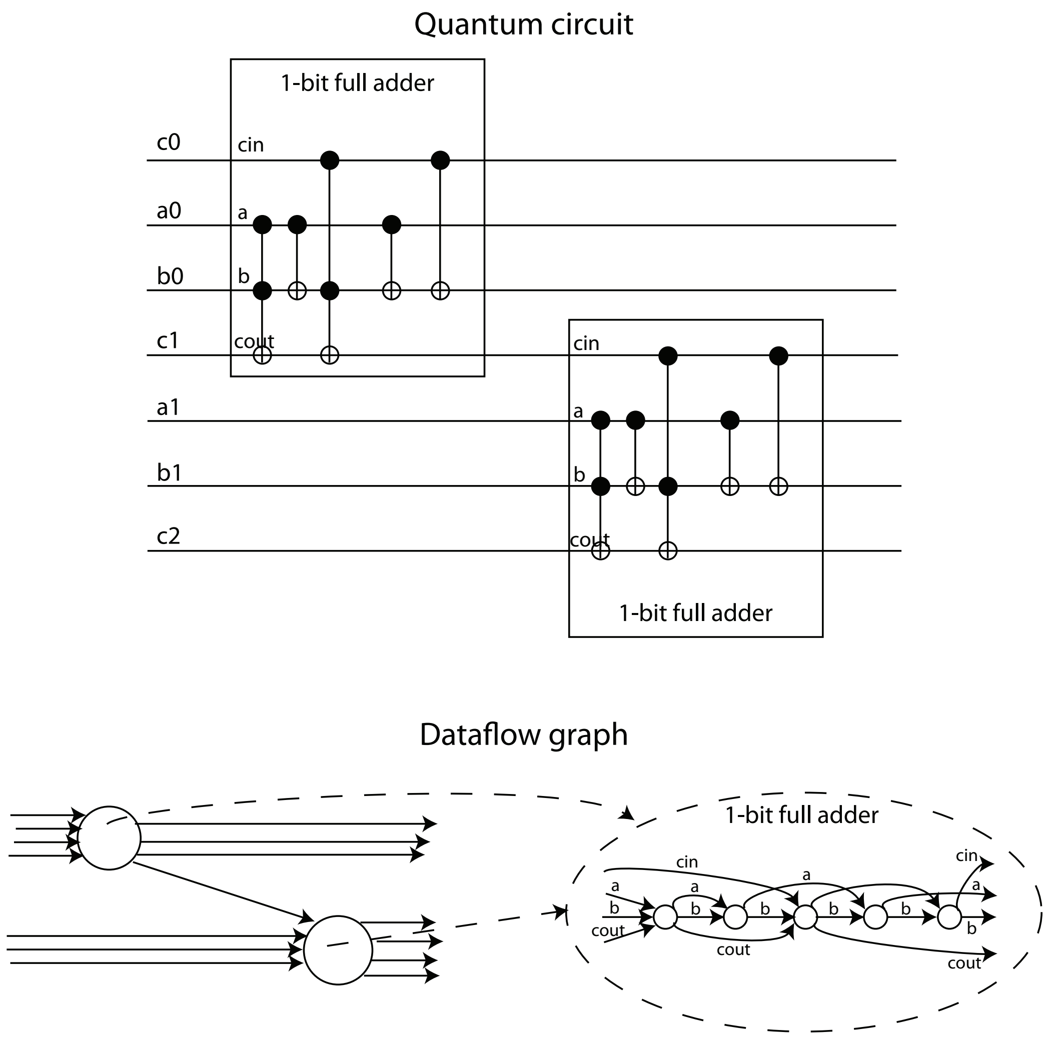


Figure 2.4: Gate networks are represented as linked, modular dataflow graphs. In this example, the top level graph consists of two nodes that each correspond to a 1-bit full adder. They both refer to the 1-bit full adder module dataflow graph.

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.

Application Dataflow Graph

Our core data structure representing the input application logic is a hierarchical, annotated dataflow graph. Figure 2.4 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. A modular dataflow graph will be used to represent it.

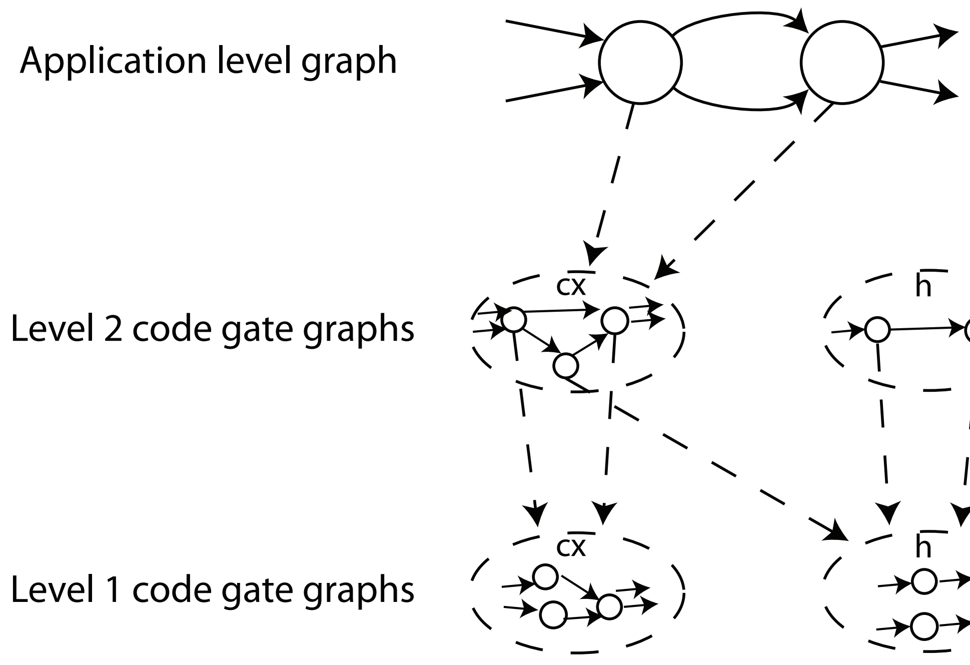


Figure 2.5: Hierarchical dataflow graphs are used to represented different levels of QEC encodings. In this example we have the 2 gate application circuit encoded in 2 levels of codes. Each code has a library of graphs, each graph implementing an encoded version of one gate type.

We maintain the hierarchy of the dataflow graph through all of the various stages of the CAD flow. For instance, when we are encoding a quantum circuit in a 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 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 2.5. 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 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 level 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 re-traversal of subgraphs adds some complexity to the traversal of our modular graphs.

2.4 Quantum Logic Synthesis

As mentioned in Section 2.3, 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.

2.4.1 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. This corresponds to the Tech-Mapping box in Figure 2.2, or more specifically to the Tech-Specific Gates portion of that step.

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.

2.4.2 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 [13].

The selection of QECC to be used in the synthesized circuit is current user-selected.

2.4.3 Error Correction Circuit Optimization

In the previous section, we discussed the placement of error correction steps in a quantum circuit. The ratio of the number of gates present in an error correction step for a common 7-bit Steane code to the number of gates in an encoded CNOT gate is about 500/7. Thus, the majority of the gates being performed in any given circuit are for error correction instead of performing the actual computation. A few other works have addressed this apparent inefficiency:

Compressed Quantum Logic Array Thaker et. al. [19] proposed converting encoded qubits between different codes depending on the frequency of operations performed on it. They proposed a memory-CPU structure where qubits that were idle in memory were stored in a stronger code and qubits undergoing computation were stored in a mixture of the same strong code and a weak code. Their reasoning was that qubits in memory required fewer corrections since they were not subject to error prone gate operations so it was less expensive to store these qubits in this code in terms of gate count. Some qubits undergoing computation would then be switched to a more lightweight code to facilitate faster computation, since both the encoded gates and the correction steps would be faster. The authors did not investigate the opposite configuration: put the qubits undergoing computation in a stronger code because they are more prone to errors while performing gates and put the qubits in memory in a weaker code because their error rates are lower.

Ancilla Factories Our work in [34] focused on identifying the large, data-independent portion of a quantum error correcting step, the fault tolerant *ancilla generation*, in order to move this circuitry off the critical path. This ancilla generation was then aggregated in *ancilla factories* which then distributed ancilla to multiple error correction steps throughout a circuit. By batch processing error correction ancilla, we found we could drastically reduce the amount of resources necessary for a given computation.

Retiming Based QECC Optimization We note that for 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 time 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. Such an assumption is faulty 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 analogue between minimizing latency in synchronous classical circuits and minimizing failures in our quantum circuits. We use the technique of circuit retiming [40] to "recorrect" the given circuit. Based on an approximation of how error propagate in a circuit, we can more effectively distribute error correction steps throughout our circuit.

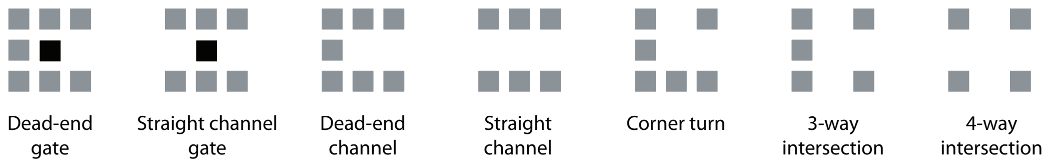


Figure 2.6: The basic building blocks of our ion trap layouts. Each *macroblock* consists of 3x3 electrodes or spaces to provide functionality as a straight channel, a gate, a turn, or an intersection.

2.5 Ion Trap Layout

Since experiments in precise ion trap layout and control are ongoing, implementation details such as electrode sizing and laser and electrode pulse sequence timings are in flux. However, we may abstract away some of these ion trap intricacies in order to get area and latency estimates for our designs. Our calculations are done using the abstraction of ion trap technology [22] described here.

Qubits A single qubit capable of holding one bit of quantum state is an ion. The physical implementation of a qubit is actually more complicated, but for our purposes, we may represent each qubit as a single ion.

Movement Electrodes are used to create potential wells in which qubits (ions) are trapped. Potential wells and the ions within are moved via an application of precise pulse sequences to the electrodes. Moving an ion around a corner takes more time than moving straight [24].

Gates A gate is performed by firing precise laser pulses at a trapped ion. We may abstract away the physics and consider that a gate is performed by arrival at certain special “gate locations” in the layout.

Macroblocks Since qubit movement is performed by electrodes whose position is fixed at fab time, certain “channels” for qubit movement are also set at fab time. The details of electrode structure are still evolving, so determining area in terms of number of ion traps is a bit ambiguous. For this reason, we use the *macroblocks* shown in Figure 2.6 as the basic building blocks of our layouts. Each macroblock has one or more “ports” through which qubits may enter and exit and which connect to an adjacent macroblock. To perform a gate operation, all involved qubits must enter a valid gate location (a black square in our macroblocks) and remain there for the duration of the gate. Our area numbers are all calculated in terms of macroblock count.

Physical Operation	Latency Symbol	Latency (μs) [21]	Error Symbol	Error Rate [20]
One-Qubit Gate	t_{1q}	1	p_{1q}	10^{-6}
Two-Qubit Gate	t_{2q}	10	p_{2q}	10^{-6}
Measurement	t_{meas}	50	p_{meas}	10^{-6}
Zero Prepare	t_{prep}	51	p_{prep}	10^{-6}
Straight Move	t_{move}	1	p_{move}	10^{-8}
Turn	t_{turn}	10	p_{turn}	10^{-8}
Idle (per μs)	N/A	1	N/A	10^{-10}

Table 2.2: Latency values and error probabilities used by our CAD flow for basic physical operations.

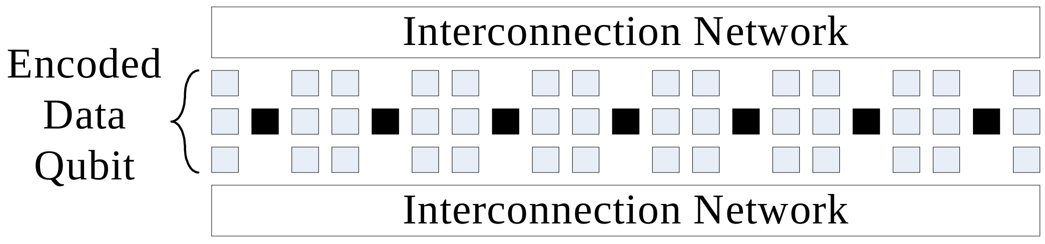


Figure 2.7: Layout of a single encoded data qubit.

We may now define latencies and error rates for ballistic movement in terms of our macroblock abstraction. Table 2.2 shows the latency values and error rates we shall be using throughout this work for both movement and for the basic quantum operations. The table also includes the symbolic notation we shall use in our equations.

2.5.1 Data Qubit Area

Over the run of a quantum circuit, encoded data must perform four distinct types of operations: transversal one-qubit gates, non-transversal one-qubit gates, transversal two-qubit gates and QEC steps. As described in Section 3.1.4, a non-transversal one-qubit gate may be performed by preparing a specific encoded ancilla and interacting it transversally with the data qubit. Likewise, the data/ancilla interaction portion of a QEC step involves a transversal two-qubit gate. In the end, the main operations the encoded data must support are transversal one- and two-qubit gates.

To support these major operations, we use single compute regions as shown in Figure 2.7. The design consists of a single column of Straight Channel Gate Macroblocks with enough room for a single encoded qubit (seven macroblocks for the [[7,1,3]] CSS code), with access on either side to whatever interconnect network is being used. Thus, if we are encoding each qubit into m physical qubits, the total area

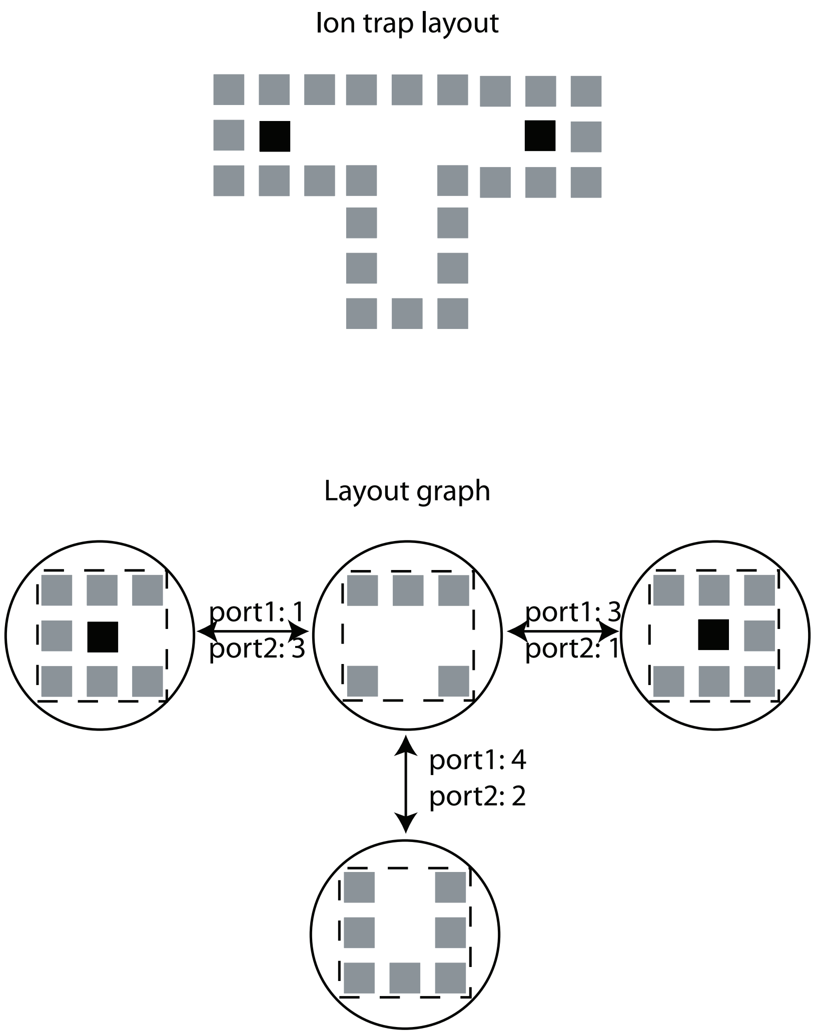


Figure 2.8: A layout and its associated graph. The nodes correspond to macroblocks and the edges correspond to "qnets" which do not have any associated physical entity but determine how macroblocks are oriented with respect to each other.

used by data is $m \times n_q$, where n_q is the total number of data qubits (including data ancillae) in the circuit.

2.5.2 Layout Graph Representation

Our layouts are represented by a layout graph which contains macroblock nodes that are linked together with QNets. The QNets hold information on how connected macroblocks are oriented with respect to one another. Figure 2.8 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 are often repeated. Some examples of repeated sublayouts are teleportation routers and ancilla factories.

Layout Specification

Our layout specification consists of a sequence of layout module instances, all parameterized 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 2.6. 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 2.9 show an example of such a modular layout.

2.6 Custom Modules

Custom modules play a two-fold part in our design flow. First, in order to facilitate hierarchical design, fully scheduled and laid out circuits may be recorded and stored for later use. For example, a 4-bit adder is a common subcircuit of larger adders, so its complete datapath and schedule may be reused by larger quantum circuits. Such modules are often available in large circuits which have clear structure and easily identifiable subcircuits.

Second, certain subcircuits are so common that they warrant hand-scheduling and optimization. The ancilla generation support infrastructure is a prime example, as QEC operations are ubiquitous in quantum computation, and the same encoded zero ancilla preparation subcircuit may be used for each QEC operation. In Chapter 3, we illustrate the process of hand-scheduling and optimizing the ancilla generation circuit. We then demonstrate the utility of this process by using our ancilla factory designs in the remainder of this work.

2.7 Fault Tolerance Verification

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

```
<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>
```

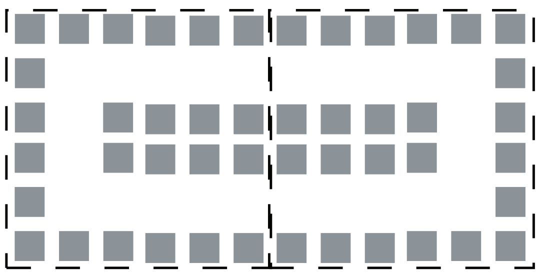


Figure 2.9: Layouts can consist of placements of single macroblocks or definition and then instantiation of larger layout blocks. In this example, we define a larger "horseshoe" block made up of macroblocks and then instantiate two of them in different positions and orientations.

computation. Furthermore, we would like to know at which points in the design are data most likely to incur errors.

2.7.1 Determining Failure Probability

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 can either track errors at the circuit level, accounting only for gate errors or at the layout level, accounting for gate, idle and movement errors.

2.7.2 Fault Tolerance Metrics

Due to the very high prevalence of errors in quantum circuits, overall probability of success will be one of the most important metrics in the foreseeable future. While reliability is only beginning to become important in classical CMOS circuits, it must be addressed as an integral design parameter from the very beginning for quantum circuits. The relatively high error rates of quantum operations motivate our inclusion of reliability as a key metric for evaluating our designs.

Since the proposed applications for a quantum computer are all currently in the complexity class NP (such as factoring, for example), we can verify whether the answer produced is correct or not fairly easily with a classical computer. This means that in we have data corrupting errors in a run of our computation, we can just run it over again until we get a correct answer. For this reason, probability of success is closely connected to the layout delay metric.

2.8 Benchmarks for Evaluating Quantum CAD

We will be using two types of benchmarks to test our heuristics. First, Shor's factorization algorithm is a practical quantum application, so we will be using the complete circuit for factorization, as well as its two primary subcircuits, an adder circuit and the Quantum Fourier Transform. Second, we will be using randomized circuits in order to test the versatility of our methods. We now elaborate on our circuit generation methodology.

2.8.1 Factorization and Its Subcircuits

For the final test of our toolset, we present in Chapter 6 our results for Shor's factorization algorithm [54]. The two chief subcircuits the factorization circuit are an adder circuit and the Quantum Fourier Transform (QFT).

We have implemented two versions of the adder circuit, one an analogue of the classical ripple-carry adder (RCA), the second an analogue of the carry lookahead adder (CLA). The quantum RCA is based on the circuits described in [16], while the quantum CLA is from [17]. The QFT is the highly parallel design presented in [26].

2.8.2 Random Circuit Generation

In addition to the real application benchmarks described above, 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 also introduce 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 is an approximate measure of the locality of communication in a circuit. It 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 sub-part 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.

2.9 ADCR-Optimal

In Section 1.3.3, we described our probabilistic area-delay product metric, ADCR. When we run a circuit through our CAD flow, what we are seeking is the *ADCR-optimal* design, or the set of design parameters which yield 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.

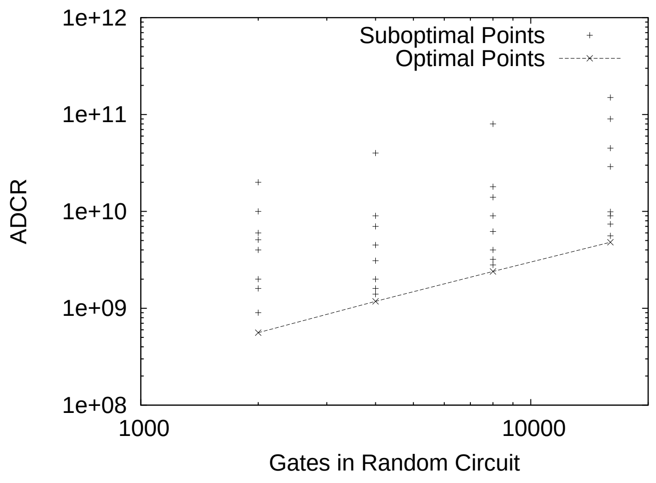


Figure 2.10: For a given input circuit, our CAD flow iterates through various datapath configurations, each of which produces a value for ADCR. The best, or *ADCR-optimal*, hardware configuration is selected.

Figure 2.10 illustrates the internal results of our toolset. For a given quantum circuit, the toolset iterates through varying hardware configurations (which will be discussed in Chapters 3 and 4), each of which results in some value of ADCR. The best configuration is automatically selected. Thus, for the remainder of this work, the ADCR values plotted in our graphs could really be considered to be ADCR-optimal values.

Chapter 3

Ancilla Factories

The bulk of both a QEC operation and a fault tolerant non-transversal gate is a preparation circuit involving the creation of encoded ancillary qubits, or *ancillae*, which does not involve the data qubit to be corrected. Consequently, as Chi et al. point out in [11], the critical path of a quantum circuit could be significantly reduced if the ancilla preparation work were done in parallel with useful computation. In particular, the speed of a quantum computation would be limited solely by *data dependencies* between encoded qubits. We refer to this fully offline parallelization of data-independent work as *running the circuit at the speed of data.*

Figure 3.1a shows a possible execution of a simple series of quantum gates involving qubits Q0, Q1 and Q2. Each gate involves some encoded ancilla preparation for the QEC step which must follow it. In addition, the non-transversal gates require specialized encoded ancilla preparation. Figure 3.1b shows these operations performed at the speed of data. Chi et al. suggest that these ancilla preparation operations could be done in advance, but the hardware cost for this parallelization grows quickly as the critical path is shortened. In this chapter, we address the need for specialized hardware to produce encoded ancillae at sufficient bandwidth.

3.1 Ancilla Preparation Circuits

In this section, we discuss several ancilla preparation circuits and evaluate them in terms of complexity and error. Ultimately, we select encoding circuits that will be used in our layouts in Section 3.3.

3.1.1 Computing on Encoded Data Bits

Since quantum data is very fragile, it must be encoded at all times in an appropriate quantum error correction code. A high-level view of the procedure for error-correcting an encoded data qubit is shown in Figure 3.2. Both the *bit value* and

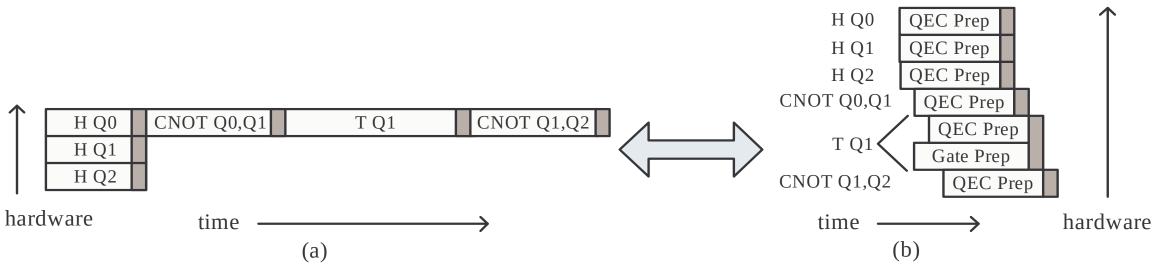


Figure 3.1: (a) Standard implementation of a circuit involving qubits Q0, Q1 and Q2. Only the grey blocks represent interactions with actual data. The bulk of the critical path involves independent ancilla preparation. (b) An optimized version of the circuit in which ancilla preparation is pulled off the critical path through use of increased hardware. Here, the speed of the computation is limited only by data dependencies (grey blocks).

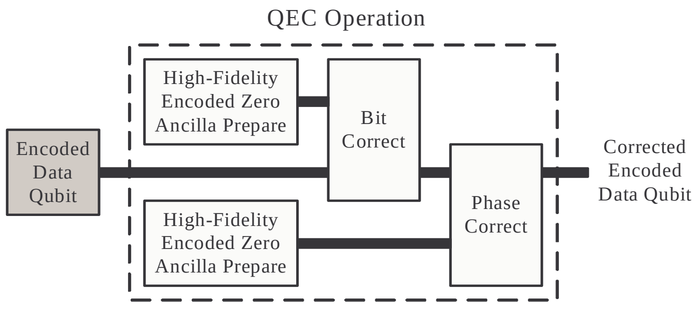

QEC Operation
High-Fidelity Encoded Zero Ancilla Prepare
Bit Correct
Encoded Data Qubit
Phase Correct
Corrected Encoded Data Qubit
High-Fidelity Encoded Zero Ancilla Prepare

Figure 3.2: A quantum error correcting (QEC) operation is composed of a *bit-flip* correction and a *phase-flip* correction, corresponding to the two types of errors that can happen to a qubit. The thick bars represent encoded qubits.

phase must be repaired during the QEC step [44]. Two sets of physical ancilla qubits are each encoded into the zero state and then consumed during correction.

Gates applied to encoded data may be classified into two types: *transversal* and *non-transversal*. A transversal encoded gate is applied by performing the corresponding physical gate *independently* on each of the qubits comprising the encoded qubit, as shown in Figure 3.3a for the Hadamard gate. A non-transversal encoded gate is decomposed into a more complex set of physical operations, including multi-qubit physical operations between physical qubits within the same encoded qubit; for example, see the Basic Encoded Zero Ancilla Prepare in Figure 3.3b. Since errors are propagated between physical qubits during the application of non-transversal gates, such gates must be designed carefully to avoid introducing uncorrectable errors.

A class of quantum codes known as CSS codes [57, 10] allow transversal implementations of most encoded gates. For this reason, CSS codes are used in most analyses

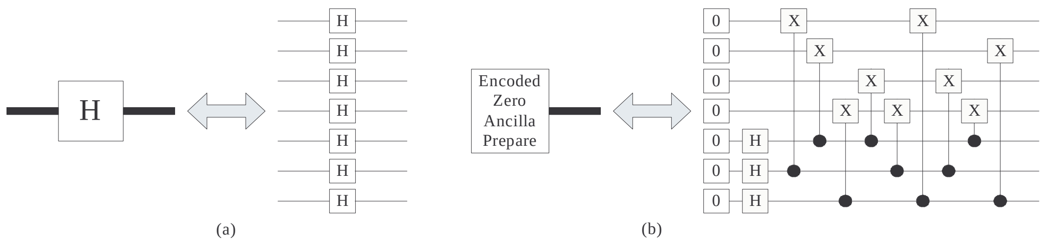


Figure 3.3: (a) A *transversal* encoded gate involves transversal application of physical gates. (b) A *non-transversal* encoded gate involves multi-qubit physical operations between physical qubits within the same encoded qubit.

of the fault tolerance of quantum circuits. Throughout this work, we use the [[7,1,3]] CSS code [57]. Encoded gates that can be performed transversally on this code include the two-qubit CX, as well as the one-qubit X, Y, Z, Phase, and Hadamard gates. In order to have a universal gate set, we also need the non-transversal $\pi/8$ gate and the encoding procedure to create an encoded ancilla. We will discuss how to obtain a fault tolerant version of the $\pi/8$ gate in Section 3.1.4.

3.1.2 Circuit Evaluation Methodology

Since encoded ancillae are a major component of error correction, it is critical to generate *clean* ancillae to avoid introducing errors during the correcting process. This ensures fault tolerance during circuit execution, as explained in Section 2.7. In the following, we will evaluate circuits by using the tools in [64] which allow us to lay out circuits. The effects of error are then modeled by Monte Carlo simulation where errors can be introduced at any gate or qubit movement operation. Additionally, we model the fact that two-qubit gates propagate bit and phase flips between qubits. This simulation is similar to what was done in [59] except with the addition of qubit movement error from our detailed layout. We assume an independent error probability for each gate and movement operation. The gate error rate is 10^{-4} and the error per movement op is 10^{-6}. Our gate and movement error rates are consistent with [60].

3.1.3 Encoded Ancilla Preparation

Since the Bit Correct and Phase Correct circuits in Figure 3.2 are fully transversal (each consisting of a transversal CX, measure and conditional correct [49]), we focus on the basic zero ancilla preparation circuit, which we introduced in Figure 3.3b. The probability of an uncorrectable error in the resulting encoded output of this circuit is 1.8×10^{-3} based on our evaluation methodology above. We would like to improve on this basic result.

There are two different circuit-level techniques for removing general errors from an

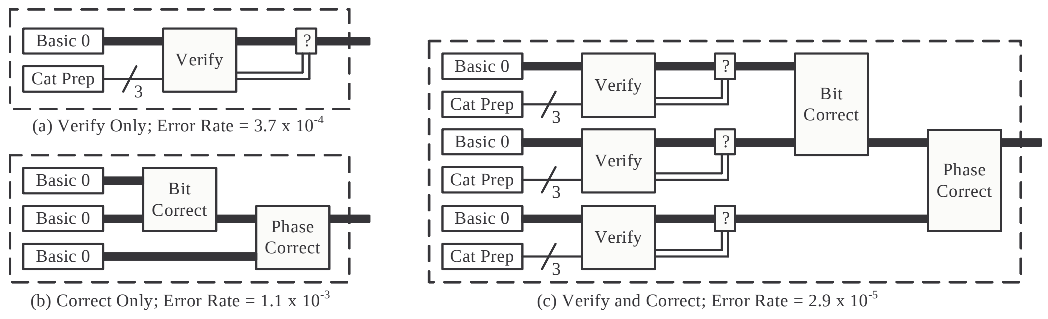


Figure 3.4: Different circuits for the "High-Fidelity Encoded Zero Ancilla Prepare" in Figure 3.2. Each "Basic 0" module corresponds to the circuit in Figure 3.3b. Each "Cat Prep" module corresponds to the preparation of a special 3-qubit state. Thick bars are encoded qubits (seven physical qubits). The overall error rate of each is given under each circuit.

encoded qubit: *verification* and *correction*. Verification tests a qubit in a known state for error and discards it if too much error is found. Correction is more complex, but it corrects a bit or phase error from an encoded qubit in an unknown state, thus it is more suitable for data qubits in a long-running computation. Encoded zero ancillae are in known state and may be discarded if necessary, so either method is suitable.

While Figure 3.3b shows the circuit for preparing an encoded ancilla in the zero state in the [[7,1,3]] CSS code, we would like a more error-free ancilla qubit for interaction with data. Figure 3.4 shows some example zero ancilla preparation circuits from the literature [61, 49], with the overall error rate for each given under the circuit. Correction alone (Figure 3.4b) loses to verification alone (Figure 3.4a) in both error and area. When comparing Figures 3.4a and 3.4c, it is important to note that they are not to scale. The "Basic 0" module (expanded in Figure 3.3b) is by far the most complex, so by doing both verification and correction, we get more than an order of magnitude improvement in error over verification alone for slightly more than three times the area. Thus, we shall use the circuit in Figure 3.4c in this paper.

Since we are using qubit verification as part of our encoded zero preparation, we need to know the success rate of verification. Using the same Monte Carlo simulation used for error probability calculations, we estimate the verification failure rate of the subunit 3.4a to be 0.2%. We will use this failure rate value in calculations later in Section 3.3.2.

3.1.4 Fault Tolerant $\pi/8$ Gate

It has been shown that *no* quantum error correcting code has transversal gate implementations for all the gates in a universal set [68], and indeed, in the [[7,1,3]] CSS code, we need the non-transversal $\pi/8$ gate in order to complete the universal set. In order to maintain fault tolerance when performing the $\pi/8$ gate on a [[7,1,3]]

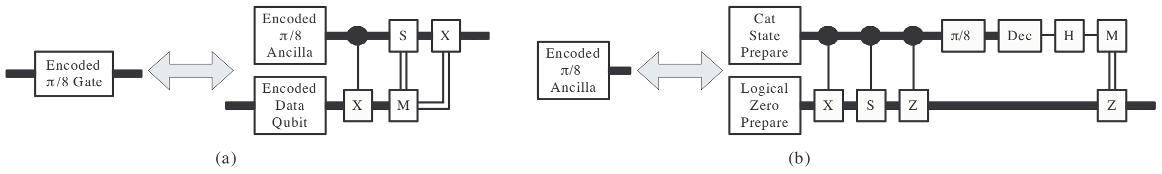


Figure 3.5: (a) Applying an encoded $\pi/8$ gate on an encoded data qubit involves creating an encoded $\pi/8$ ancilla and performing some transversal gates. (b) Creating the encoded $\pi/8$ ancilla used in the circuit in (a) requires an encoded zero ancilla, a 7-qubit cat state (a specially prepared qubit set) and a series of transversal gates. Note that the $\pi/8$ gate near the far right is transversal but does not implement an encoded $\pi/8$ gate.

encoded qubit, we use a technique developed in [69]. Their approach is to generate an encoded ancilla qubit encoded in the $\pi/8$ state and perform transversal interactions with the data, as shown in Figure 3.5a, to achieve the overall effect of an encoded $\pi/8$ gate.

To encode the $\pi/8$ ancilla qubit, we could try to create a physical $\pi/8$ ancilla qubit and then use the encoding circuit in Figure 3.3b, but this approach would result in errors on the original physical qubit propagating to each physical qubit in the final encoded ancilla, which is unacceptable. Thus, we require the far more complicated circuit shown Figure 3.5b, which consists of an encoded zero ancilla prepare, a 7-qubit cat state prepare (where a cat state is a specially prepared multi-qubit state) and a series of transversal encoded gates.

3.1.5 Fault Tolerant $\pi/2^k$ Gates

The Quantum Fourier Transform (QFT) requires controlled phase rotation gates by small angles (these gates replace the explicit tracking of roots of unity in the classical FFT algorithm). The amount of precision for these gates scales exponentially in the number of bits involved in the QFT [44]. A controlled phase rotation by $\pi/2^k$ can be generated by a CX gate and 3 single qubit $\pi/2^{k+1}$ gates [25]. Thus, using circuit techniques mentioned so far, we can implement every gate in the QFT fault tolerantly except these single qubit rotation gates. There are two problems with implementing an arbitrary precision phase rotation fault tolerantly:

- For angles smaller than $\pi/2$, there exists no transversal gate implementation using the [[7,1,3]] code [68]. In fact, this seems likely to be true for all codes.
- Such a gate would require the physical implementation of an arbitrary precision rotation – a difficult burden on the engineers of these devices.

Due to the above reasons, we adopt a technique by Fowler [25]. To approximate small angle rotations, we exhaustively search all permutations of T and H gates to find a

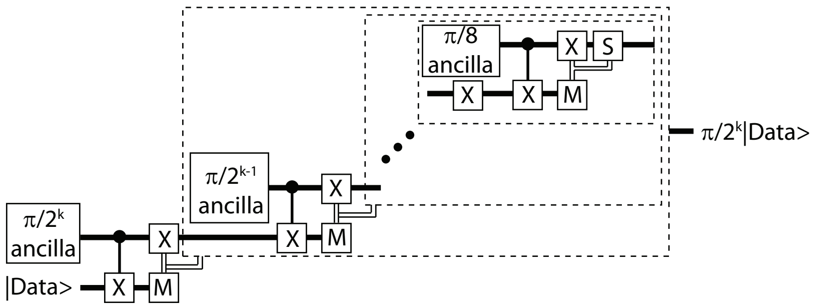


Figure 3.6: Fault tolerant $\pi/2^k$ gates can be performed recursively with a cascade of $\pi/2^i|i = 3...k$ ancilla factories and $k-2$ CX and X gates. Each measure gate output controls both the single qubit X gate and the compound gate involving more ancilla factories. Each measurement has a equal chance of giving the "correct" state, in which the remaining circuit is skipped or a "wrong" state in which a larger rotation has to be done to adjust the state. The actual output data from the circuit connects to the first quantum bitline associated with a correct measurement.

minimum length sequence for a $\pi/2^k$ rotation gate up to an acceptable error.

We also note that if a $\pi/2^k$ physical gate is available in a given technology, an exact fault-tolerant $\pi/2^k$ can be implemented as shown in Figure 3.6. In order to be conservative about the availability of arbitrary precision rotation gates, we do not use this construction in the circuits in this paper. However, in Section 3.3.3, we briefly analyze the performance advantages of this technique.

3.2 Benchmark Characteristics: Ancilla Need

We now characterize the runtime properties of some commonly used quantum circuits, focusing on the impact of encoded ancilla generation. Many quantum algorithms require ancillae to assist in computation. For example, an n-bit Quantum Ripple-Carry Adder uses two n-bit data inputs plus $n+1$ ancillae. In addition, shorter-lived ancillae are needed for QEC and for performing non-transversal encoded gates, as discussed earlier.

Throughout this work, we refer to the longer-lived ancillae used in the main computation as "data ancillae" and to the shorter-lived ones as "ancillae." We make this distinction because data ancillae tend to have long enough lifespans that "discarding" them and restarting their portion of the computation has a relatively high cost. Our work focuses on the short-lived ancillae which need to be produced in large quantities and which may more easily be discarded and re-encoded.

We do most of our analysis in a symbolic fashion so that it may be applied to varying technologies and assumptions. However, we will also be applying the analysis to a specific technology, trapped ions [22], in order to make the results of our calculations

Circuit	Qubit Count	Gate Count
32-bit QRCA [16]	97	2177
32-bit QCLA [17]	123	2410
32-bit QFT [44]	32	8145

Table 3.1: Total encoded qubit count and total encoded gate count for our benchmarks, the 32-bit Quantum Ripple-Carry Adder (QRCA), the 32-bit Quantum Carry-Lookahead Adder (QCLA) and the 32-bit Quantum Fourier Transform (QFT).

more concrete. We use the physical gate latencies shown in Table 2.2, the [[7,1,3]] CSS code introduced in Section 3.1.1 and the encoded ancilla preparation circuits shown in Figures 3.4c and 3.5b. Note that the "Zero Prepare" in Table 2.2 refers to a physical zero prepare, which is the leftmost set of gates in the Basic Encoded Zero Ancilla Prepare in Figure 3.3b.

In the next section, we discuss the three benchmarks we have chosen to study. Then in the following section, we characterize various properties of these quantum circuits, which shall aid us in making design decisions in the remainder of our work.

3.2.1 Benchmarks

For our benchmarks, we use the 32-bit Quantum Ripple-Carry Adder (QRCA) circuit from [16], the 32-bit Quantum Carry-Lookahead Adder (QCLA) circuit from [17] and a 32-bit Quantum Fourier Transform (QFT) circuit we derived using methodology described in Section 3.1.5. All three are core kernels of a varied array of quantum algorithms, including Shor's factorization algorithm. Table 3.1 summarizes the size of each circuit.

In addition to their common use, these circuits have been chosen because they run the gamut from largely serial to highly parallel. All three are similar to their classical counterparts in terms of parallelism. The QRCA is a rather serial circuit, the QCLA is somewhat more parallel and the QFT is maximally parallel, much like the classical Fast Fourier Transform.

3.2.2 QEC Circuit Characteristics

We study our benchmark circuits at two extremes of the latency-area trade-off: 1) No overlap of QEC and computation (high latency, but low area), and 2) infinitely fast encoded ancilla production, resulting in an execution limited only by data dependencies (low latency, but potentially much higher area for encoded ancilla generation).

Table 3.2 shows for each benchmark the latency of the critical path in the absence of movement (Column 2), as well as latencies for the data-dependent and data-

Circuit	Data Op Latency (μs) (% of total)	Data QEC Interact Latency (μs) (% of total)	Ancilla Prep Latency (μs) (% of total)
32-Bit QRCA	29508 (5.2%)	95641 (16.7%)	447726 (78.2%)
32-Bit QCLA	3827 (5.3%)	11921 (16.7%)	55806 (78.0%)
32-Bit QFT	77057 (5.0%)	365792 (23.7%)	1097376 (71.2%)

Table 3.2: Relative latency of useful data operations, interaction of data with encoded ancillae for QEC and encoded ancilla preparation for QEC for various circuits, assuming no overlap between them.

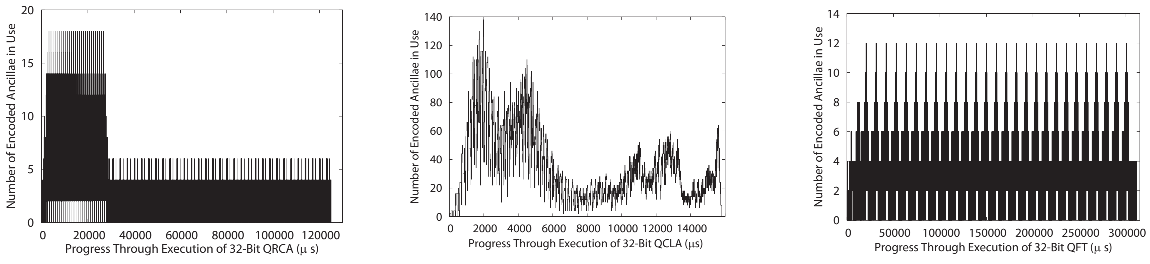


Figure 3.7: Encoded zero ancilla needs for the QRCA (left), the QCLA (middle) and QFT (right) to run at the speed of data.

independent (Columns 3 and 4) portions of QEC steps, assuming a QEC operation must be performed after each useful gate. The minimal running time is the sum of Columns 2 and 3, since these involve data qubits. Column 4 corresponds to encoded ancilla generation time. Clearly, there is much to be gained in overall execution time by taking ancilla preparation off the critical path.

Figure 3.7 shows the number of encoded ancillae consumed as a function of time for QEC in order to keep the circuit operating at the speed of data, which means that adequate hardware resources exist to generate and distribute the needed ancillae in

Circuit	Avg Bandwidth Needed for QEC (Encoded Zero Ancillae / ms)	Avg Bandwidth Needed for Non-Transversal Gates (Encoded $\pi/8$ Ancillae / ms)
32-Bit QRCA	34.8	7.0
32-Bit QCLA	306.1	62.7
32-Bit QFT	36.8	8.6

Table 3.3: Average bandwidth of encoded zero ancillae needed for QEC and average bandwidth of encoded $\pi/8$ ancillae needed for non-transversal one-qubit gates if each circuit is to be executed at the speed of data. Note that bandwidth is given per *milli*second.

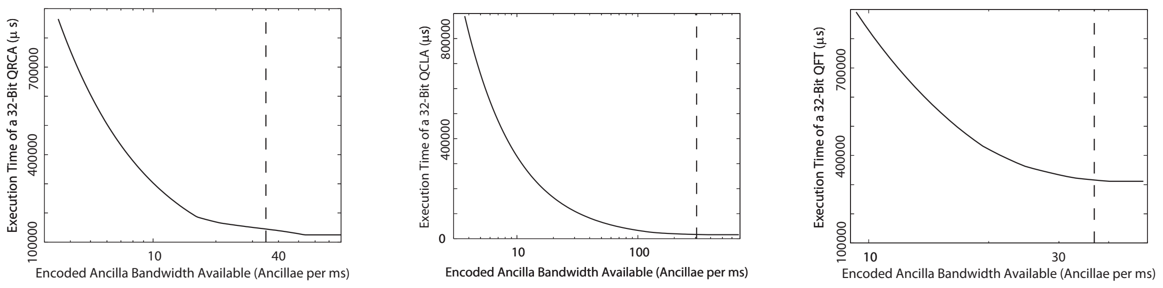


Figure 3.8: The execution time of the QRCA (left), the QCLA (middle) and the QFT (right) as a function of a steady throughput of encoded zero ancillae. The vertical line in each shows the average bandwidth for that circuit from Table 3.3.

time, but the interaction with data during each QEC step is still on the critical path of execution. Table 3.3 summarizes this figure by giving the average encoded ancilla bandwidth necessary for each.

These averages do not take into account the handling of peak periods. In reality, the encoded ancilla bandwidth necessary to run a circuit optimally may be higher than the average bandwidth. Figure 3.8 shows for our benchmarks the circuit execution time assuming a steady throughput of encoded ancillae being generated, as specified on the x-axis. These graphs show us the sustained ancilla bandwidth necessary to run each circuit at near-optimal speed, but these are only estimates since they lack the details of movement and layout. In Section 3.3, we examine the associated hardware needs.

3.2.3 Non-Transversal One-Qubit Gates

The encoded ancilla bandwidth discussed in Section 3.2.2 included only zero ancillae needed for error correction. Non-transversal one-qubit gates account for 40.5%, 41.0% and 46.9% of our QRCA, QCLA and QFT benchmarks circuits, respectively, when using the [[7,1,3]] encoding. As explained in Section 3.1.4, the execution of a non-transversal encoded gate is performed with the use of a $\pi/8$ encoded ancilla qubit. Column 3 in Table 3.3 shows the corresponding $\pi/8$ ancilla bandwidth needed for each benchmark to achieve a runtime limited only by the speed of data (the sum of Columns 2 and 3 in Table 3.2).

3.3 Ancilla Factory Layout

In this section, we shall explore the design space of possible ancilla factories and determine the hardware resources necessary to produce encoded ancillae at the bandwidths calculated in Sections 3.2.2 and 3.2.3 in order to take ancilla generation off

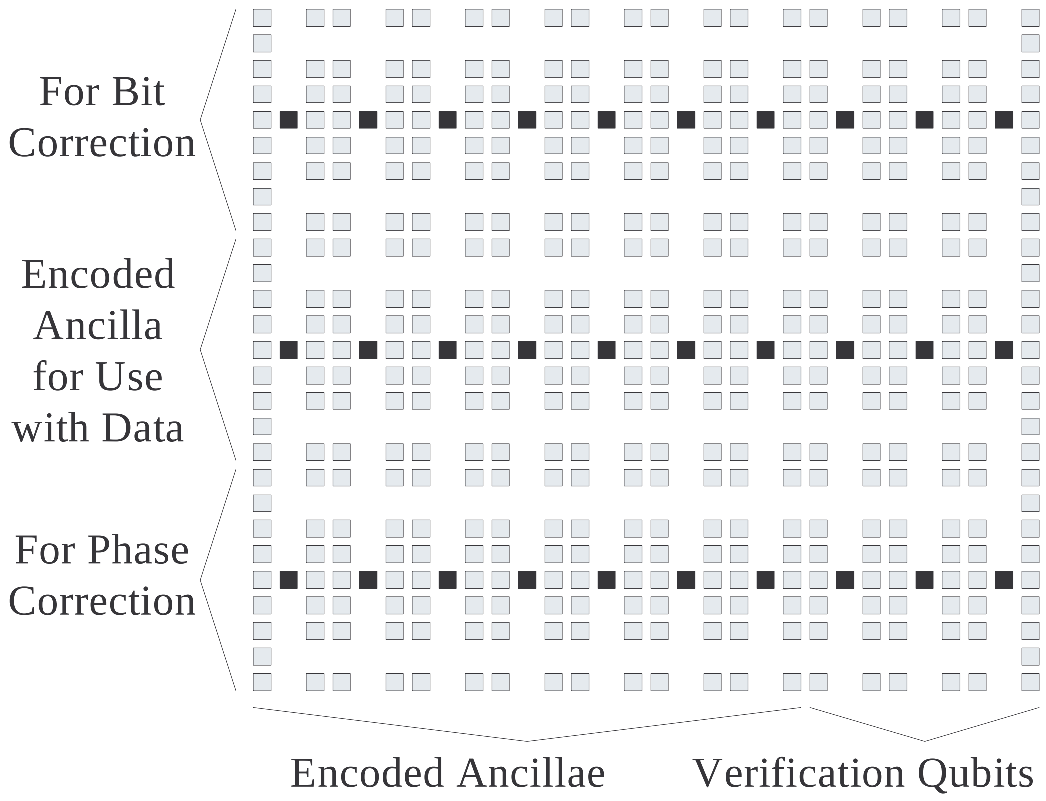


Figure 3.9: An ancilla factory for the circuit in Figure 3.4c. Each row of gates generates and verifies one of the three encoded zero ancillae, then bit and phase correction are performed.

the critical path of execution. We will use the ion trap macroblock layout scheme introduced in Section 2.5.

3.3.1 Simple Ancilla Factories

We now focus on designing an *ancilla factory*, a concept first proposed in [58]. An ancilla factory is a portion of the layout which consumes stateless physical ancillae and produces a steady stream of encoded ancillae at some rate. Figure 3.9 shows a simple ancilla factory to execute the circuit in Figure 3.4c. Each row of gates has room for ten physical qubits, seven to be encoded and three for verification. The adjacent rows are used for communicating. When all three are encoded and verified, the middle one is bit-corrected by the top one and phase-corrected by the bottom one. Using a hand-optimized schedule, the total latency of a single ancilla preparation is approximately: $t_{prep} + 2 \times t_{meas} + 6 \times t_{2q} + 2 \times t_{1q} + 8 \times t_{turn} + 30 \times t_{move}$.

Substituting in the ion trap latencies from Table 2.2, the layout in Figure 3.9 has a total latency of 323μs with a throughput of 3.1 encoded ancillae per millisecond

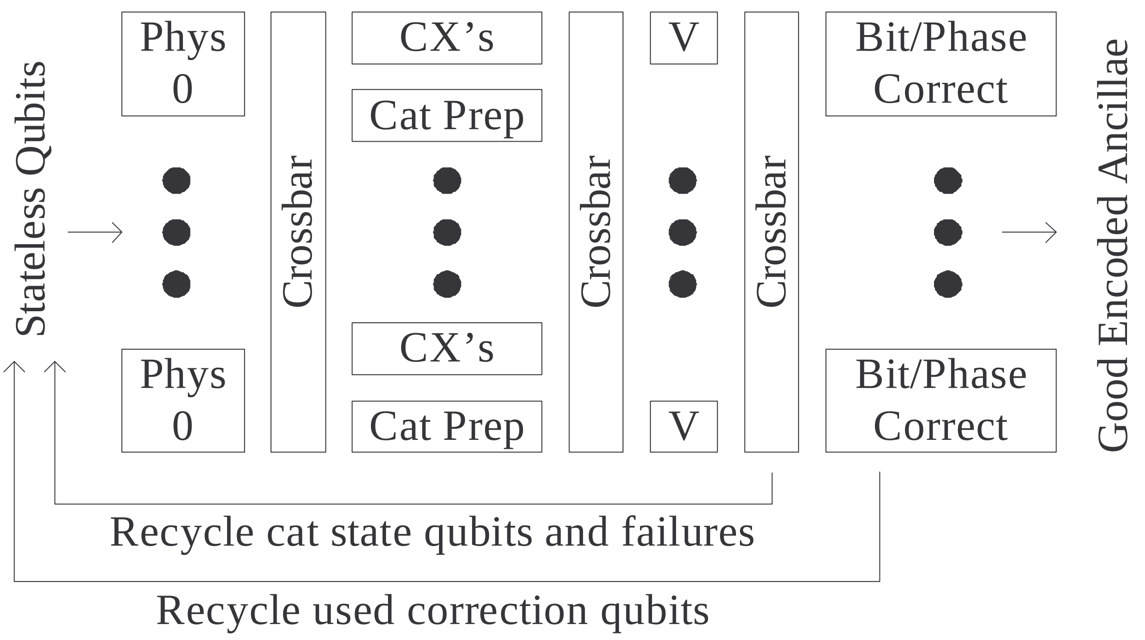


Figure 3.10: A fully pipelined encoded zero ancilla creation unit implementing the circuit in Figure 3.4c.

and an area of 90 macroblocks. Using this simple ancilla factory, we could produce any desired bandwidth of encoded ancillae by replicating the layout as many times as necessary. Unfortunately this design is inefficient in that the verification qubits needlessly take up space during the seven-qubit zero encoding procedure. To combat this inefficiency we instead consider a pipelined approach.

3.3.2 Encoded Zero Ancilla Factory

Classically, pipelining a circuit is done by inserting synchronization points (registers) into the circuit's datapath to enable logic reuse, thereby increasing throughput with a small increase in latency. We can apply a similar technique to our ancilla factory layout in an effort to improve area utilization. Due to the precise electrode and laser pulse sequences needed to implement movement and gates, ion trap layouts are by definition synchronous without additional synchronization elements. Instead, we must add a set of communication channels between pipeline stages allowing qubit movement for maximum gate location occupancy.

We consider the entire circuit for fault tolerant encoded zero ancilla creation (Figure 3.4c). Figure 3.10 shows a fully pipelined microarchitecture for this circuit, which consists of four stages. Each stage contains a number of functional units for its subcircuit such that the output bandwidth of one stage is matched to the input bandwidth of the next. Adjacent stages are separated by a crossbar (Figure 3.11a), which con-

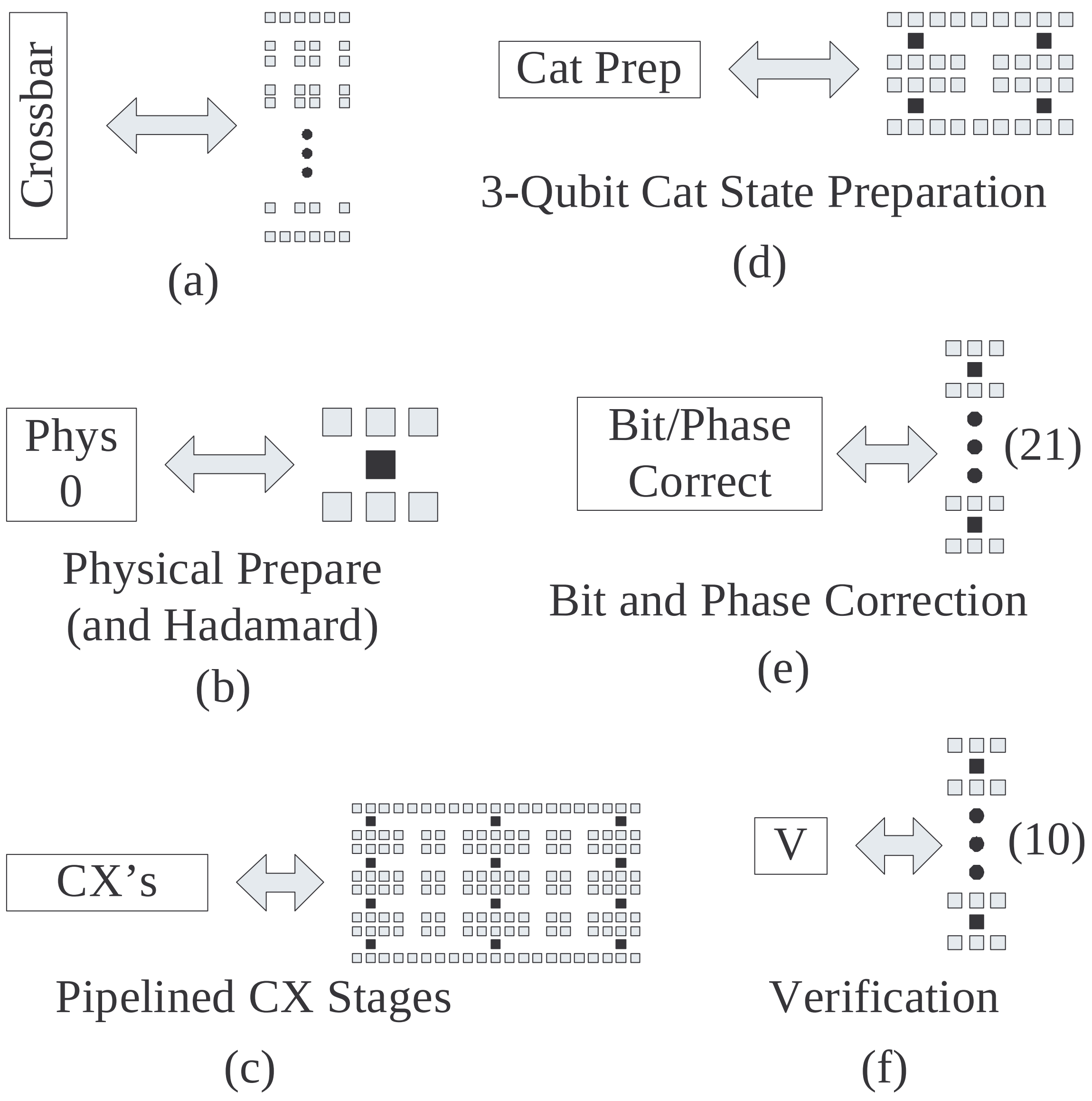


Figure 3.11: A layout of each unit in Figure 3.10.

Functional Unit	Symbolic Latency
Zero Prep	$t_{prep} + t_{1q} + 2 \times t_{turn} + t_{move}$
CX Stage	$3 \times t_{2q} + 6 \times t_{turn} + 5 \times t_{move}$
Cat State Prep	$2 \times t_{2q} + 4 \times t_{turn} + 2 \times t_{move}$
Verification	$t_{meas} + t_{2q} + 2 \times t_{turn} + 2 \times t_{move}$
B/P Correction	$t_{meas} + 2 \times t_{2q} + 6 \times t_{turn} + 8 \times t_{move}$

Table 3.4: The symbolic latency for each functional unit in Figure 3.10.

sists of two vertical columns, fully connected horizontally, one for upwards movement, the other for downwards. In this way, each qubit may move from one stage to the next without worrying about congestion, since qubits moving in opposite directions won't impede each other.

Stage 1 consists of preparing a junk physical qubit into the zero state with an optional Hadamard gate at a single gate location (Figure 3.11b). Even though only some of these qubits need the Hadamard, we group them all into the same set of functional units.

Stage 2 consists of two types of units. Looking at the CX portion of the ancilla prepare circuit in Figure 3.3b, we see that the first three CX's can be performed in parallel, as can the next three, followed by the final three. Thus, we may use the pipelined layout in Figure 3.11c for this functional unit, with three sets of qubits (each performing three CX's with one idle qubit) in this functional unit at a time. The Cat Prep units (Figure 3.11d) create a three-qubit cat state out of three physical zero ancillae by performing two CX's in succession.

Verification of the encoded zero ancillae using the cat states is performed in Stage 3 and involves performing three CX's in parallel and then measuring the cat state qubits to determine success or failure of the encoded ancilla. Since the encoded ancilla qubits must wait for the measurement to complete, we need 10 macroblocks, one for each qubit as shown in Figure 3.11e. When this stage is complete, the three qubits of the cat state are recycled immediately, as well as the other seven qubits if the verification failed.

Finally, in Stage 4, a verified encoded zero ancilla A is first bit-corrected by a verified encoded zero ancilla B and then phase-corrected by a verified encoded zero ancilla C. Since we need storage room for A plus room to measure both B and C in parallel (allowing us to overlap these measurements in time), each such functional unit needs space for three encoded ancillae, as shown in Figure 3.11f.

Table 3.4 shows the symbolic latency breakdown for each stage of the pipeline, while Table 3.5 shows various numeric characteristics for each functional unit under our ion trap assumptions. Note that Stages 3 and 4 have input bandwidth different from output bandwidth due to the fact that some qubits are used up and recycled in these stages. To achieve high resource utilization, we determine unit count by matching bandwidth between successive stages. The results are shown Table 3.6.

Functional Unit	Latency (μs)	Stages	BW (qubits/ms) In	BW (qubits/ms) Out	Area
Zero Prep	73	1	13.7	13.7	1
CX Stage	95	3	221.1	221.1	28
Cat State Prep	62	2	96.8	96.8	6
Verification	82	1	122.0	85.2	10
B/P Correction	138	1	152.2	50.7	21

Table 3.5: The numeric characteristics for each functional unit in Figure 3.10 using our ion trap assumptions. "Stages" is the number of pipeline stages within the functional unit itself, and "Area" is given in number of macroblocks.

Functional Unit	Unit Count	Total Height	Total Area
Zero Prepare	24	24	24
CX Stage	1	4	28
Cat State Prepare	1	2	6
Verification	3	30	30
B/P Correction	2	42	42

Table 3.6: The functional unit counts and stage characteristics for the encoded zero ancilla factory in Figure 3.10. The CX and Cat Prepare units in Stage 2 are bandwidth matched to a ratio of 7 to 3 (which is appropriate for verification), and then the other stages are sized to match the resultant bandwidth.

For the crossbars, we use a two-column design, one column for upwards movement, the other for downwards, in order to avoid congestion. However, physical qubits exiting Stage 1 are funneled inward to the much smaller Stage 2, so we use a single column crossbar since bi-directionality is likely unnecessary. The total crossbar area is thus 24 + 2 * 30 + 2 * 42 = 168 macroblocks, and the total functional unit area is 24 + 34 + 30 + 42 = 130 macroblocks, resulting in a total area of 298 macroblocks.

For overall throughput, we take the minimum throughput among the stages. The bottleneck in the factory is the CX Stage. Each seven physical qubits out of this stage correspond to an encoded zero ancilla. Approximately 99.8% of these qubits are successfully verified (using the results of our Monte Carlo simulations mentioned in Section 3.1.3), and two-thirds of them are then used to correct the other third. Thus, the overall throughput of our zero ancilla factory is: $\frac{221.1}{7} \times 0.998 \times \frac{1}{3} = 10.5$ encoded ancillae / ms.

Stage	Symbolic Latency
Cat State Prepare	$7 \times t_{2q} + 14 \times t_{turn} + 8 \times t_{move}$
Transversal CX/CS/CZ/$\pi/8$	$3 \times t_{2q} + 2 \times t_{turn} + 3 \times t_{move}$
Decode (plus Store)	$7 \times t_{2q} + 14 \times t_{turn} + 8 \times t_{move}$
H/M/Transversal Z	$t_{meas} + 2 \times t_{1q} + 2 \times t_{turn} + 2 \times t_{move}$

Table 3.7: The symbolic latency for each stage in the encoded $\pi/8$ ancilla generation circuit.

Stage	Latency	In BW	Out BW	Area
Cat State Prepare	218	32.1	32.1	12
Transversal CX/CS/CZ/$\pi/8$	53	264.2	264.2	7
Decode (plus Store)	218	64.2	36.7	19
H/M/Transversal Z	74	108.1	94.6	8

Table 3.8: The numeric characteristics for each stage in the encoded $\pi/8$ ancilla generation circuit under our ion trap assumptions.

3.3.3 Encoded $\pi/8$ Ancilla Factory

In Section 3.2.3, we showed that a non-trivial supply of encoded $\pi/8$ ancillae are also needed by our circuits. The circuit in Figure 3.5b shows how to turn a zero ancilla generated by our pipelined ancilla factories into an encoded $\pi/8$ ancilla. We pipeline this circuit as well, not merely for the benefit of input/output ports, but also because the different number of qubits involved at various points in the circuit means that resources would be idle if this were done "in place." This circuit may be divided into four stages: 1) Cat State Prepare, 2) Transversal Controlled-Z/S/X, plus Transversal $\pi/8$, 3) Decode, 4) One-qubit H, One-qubit Measure, Transversal Z conditional on measurement.

Table 3.7 shows the symbolic characteristics of each of these stages, while Table 3.8 shows the numeric values. Note that bandwidths here are in physical qubits, which is why Stages 1 and 3 have differing bandwidths despite having the same latency. We

	Unit	Total	Total
Stage	Count	Height	Area
Cat State Prepare	4	24	48
Transversal CX/CS/CZ/$\pi/8$	1	7	7
Decode (plus Store)	4	52	76
H/M/Transversal Z	2	16	16

Table 3.9: The functional unit counts and characteristics for each stage of our final $\pi/8$ ancilla factory.

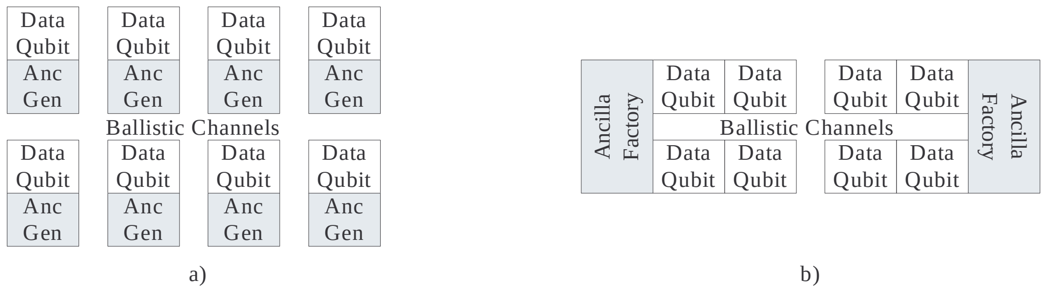


Figure 3.12: a) Ancilla production proposed in prior work: Each data qubit location is adjacent to an In-Place Ancilla Generator which produces encoded ancillae exclusively for that location. b) Multiplexed Ancilla Factories: Data qubits are more tightly packed together, with ports to Ancilla Factories at the edges of the Compute Region. (not to scale)

now match bandwidths just as we did for the zero ancilla factory in order to get close to full utilization. Table 3.9 shows the the final unit counts of our $\pi/8$ ancilla factory. Note that only half the qubits consumed by Stage 2 come from Stage 1 (the others come from an encoded zero ancilla factory).

The total stage heights are different enough that an exact layout would likely require partially folding some stages into others and simulating execution to determine exact crossbar sizes needed to avoid congestion. For our purposes, we will allocate two columns to each crossbar, since qubits must be able to move in both directions at the same time. Thus, the total crossbar area is $2 * 24 + 2 * 52 + 2 * 52 = 256$ macroblocks, and the total functional unit area is $48 + 7 + 76 + 16 = 147$ macroblocks, resulting in a total area of 403 macroblocks. We note, however, that this is merely the area for turning an encoded zero into an encoded $\pi/8$. This factory needs to be supplied by zero ancilla factories in order to function, which we account for in Section 3.4.

The bottleneck of this ancilla factory is the Cat State Prepare stage. Each seven-qubit cat state produced by this stage results in one encoded $\pi/8$ ancilla produced by the factory, so the throughput of the factory is equal to the throughput of this stage: 18.3 encoded $\pi/8$ ancillae / ms.

As mentioned in Section 3.1.5, we build up smaller angle $\pi/2^k$ rotations from combinations of $\pi/8$ and H gates instead of building ancilla factories for them. It is worthwhile to note that if physical gates with adequate precision are available, the critical path for the data can be decreased further. From Figure 3.6 we see that the critical path for the data through such a factory would on average consist of $\sum_{i=0}^{k-2} 1/2^k$ CX gates and one fewer X gates.

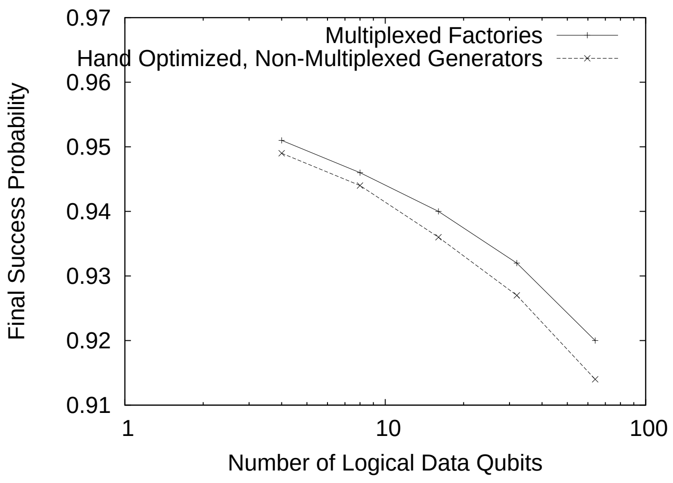


Figure 3.13: By bringing data qubits closer together within a Compute Region and confining multiplexed ancilla factories to the edges, the multiplexed approach results in slightly better success probability than the hand optimized, one ancilla generator per data qubit approach.

3.3.4 Qubit Fidelity When Multiplexing

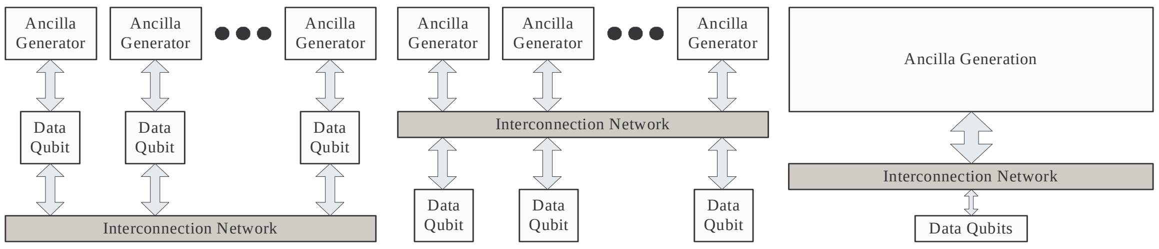
In [23], the authors propose ancilla production as illustrated in Figure 3.12a. Each data qubit location has an In-Place Ancilla Generator next to it. All necessary ancillae for that location must be produced by that Generator, as there is no multiplexing. In [39], the authors optimize this approach by carefully hand scheduling all ballistic movement for a QEC operation, including interaction with the data.

Figure 3.12b shows our approach introduced in this chapter, wherein the data are more tightly packed to reduce distances between data in a single Compute Region. Pipelined, multiplexed ancilla factories are placed at the edges of the Compute Region, then encoded ancillae are brought to the data locations as needed.

We have already illustrated the area and latency benefits of our approach. Multiplexing reduces wasted area as ancilla factories may be sized according to the ancilla need of a target circuit, thus minimizing idle ancilla generation hardware. Likewise, in the former approach, imbalanced gate location use results in execution stalls due to a few overburdened ancilla generators, while many others remain idle. With multiplexed factories, execution is less likely to stall since the load on the ancilla generation resources is more evenly distributed.

However, qubit fidelity is a potential drawback when using the latter approach. The data qubits are closer together and thus theoretically don't need to move as far ballistically. However, the encoded ancillae generated at the output ports of the factories now need to move further to reach the data.

We evaluate this trade-off by running random graphs of 100,000 logical gates using



(a) QLA/CQLA Microarchitecture (b) Fully-Multiplexed Ancilla Distribution (c) Microarchitecture in (b) to scale for the 32-bit QCLA

Figure 3.14: A quantum layout microarchitecture may be considered to consist of three components: generators of encoded ancillae, data qubit computation regions and interconnect. (a) The (C)QLA microarchitecture dedicates an ancilla generation unit to each data qubit. (b) Our general microarchitecture redirects encoded ancillae to wherever they're needed on the chip, thus avoiding idle generators. (c) In order to run at the speed of data, the ancilla generation portion of the chip needs far more hardware than the data regions, as shown in Table 3.10.

the two different approaches. In each test, the datapath is a single Compute Region with just enough gate locations to fit the data qubits. Figure 3.13 shows the results for varying numbers of data qubits. The benefit of bringing the data qubits in closer proximity to each other outweighs the detriment of having to bring encoded ancillae to data in the non-multiplexed approach. Thus, we will henceforth use multiplexed, pipelined ancilla factories in our designs.

3.4 Architectural Trade-offs

We now bring our analyses together to draw quantitative conclusions about running a quantum circuit at the speed of data and to compare against proposed architectures from prior work. Following that, we present a more qualitative discussion of some conclusions we've drawn from this work.

3.4.1 Matching Production to Need

We divide the microarchitecture of a quantum layout into three components: 1) hardware resources for generation of encoded ancillae; 2) hardware resources for data operations, including operations involving data ancillae and the data/ancilla interaction portion of a QEC step; and 3) an interconnection network for moving around both encoded data and ancillae. Figure 3.14a shows the (C)QLA microarchitecture [23, 19] using these components, with each data qubit (whether in a compute region or memory) having an associated ancilla generation unit for QEC. Figure 3.14b shows an ancilla factory-based microarchitecture wherein encoded ancillae are being gener-

Quantum Circuit	Data Area (% of total)	QEC Ancilla Factories Area (% of total)	$\pi/8$ Ancilla Factories Area (% of total)
32-Bit QRCA	679 (33.6%)	986.9 (48.8%)	354.7 (17.6%)
32-Bit QCLA	861 (6.8%)	8682.2 (68.4%)	3154.4 (24.8%)
32-Bit QFT	224 (13.2%)	1043.5 (61.3%)	433.7 (25.5%)

Table 3.10: Area breakdown to generate encoded ancillae at the QEC bandwidths shown in Table 3.3. The $\pi/8$ ancilla bandwidth is computed to match. The last column includes area for both $\pi/8$ encoding and the zero ancilla factories supplying these encoders.

ated across the chip and distributed to data as need dictates. For these results, we assume that a QEC step is performed after each data qubit operation.

Table 3.10 gives the relative areas of two of the three components of the microarchitecture in Figure 3.14b when running our benchmarks at (or near) the speed of data under our ion trap assumptions. We depict our microarchitectural components to scale for the 32-bit QCLA in Figure 3.14c. The encoded zero ancilla bandwidth for error correction is the average bandwidth required for each circuit (Table 3.3). A corresponding encoded $\pi/8$ ancilla bandwidth is computed (but not shown in the table) to run the circuit at that speed. Column 4 includes only those zero ancilla factories producing for QEC. Column 5 includes both $\pi/8$ encoding factories and sufficient encoded zero ancilla factories to supply the $\pi/8$ encoding factories.

We see that even the most serial of the benchmarks, the Quantum Ripple-Carry Adder, requires a substantial portion of the chip (two-thirds) dedicated to encoded ancilla generation in order to take this generation off the execution's critical path, while the more parallel QCLA requires more than 90%.

3.4.2 Latency/Area Evaluation

The proposals for both QLA and CQLA specify space for only serial production of ancillae at each encoded data qubit location. We generalize these architectures to GQLA and GCQLA, in which we replicate the ancilla area at each data qubit to allow parallel production of ancillae. CQLA has additional flexibility in that different numbers of data units can be present in the compute region. We wish to quantify the efficiency of ancilla production in each microarchitecture by studying area needed for a given execution time.

Methodology: Using dataflow graphs of our benchmarks and the estimates in Tables 3.4-3.9, we implemented an event-based simulation of ancilla factory production and data qubit gate consumption. Simulation of the QLA [23] microarchitecture assumes that each data qubit in the computation has a dedicated cell with ancilla production. Data qubits are always moved back to their home base to do the error

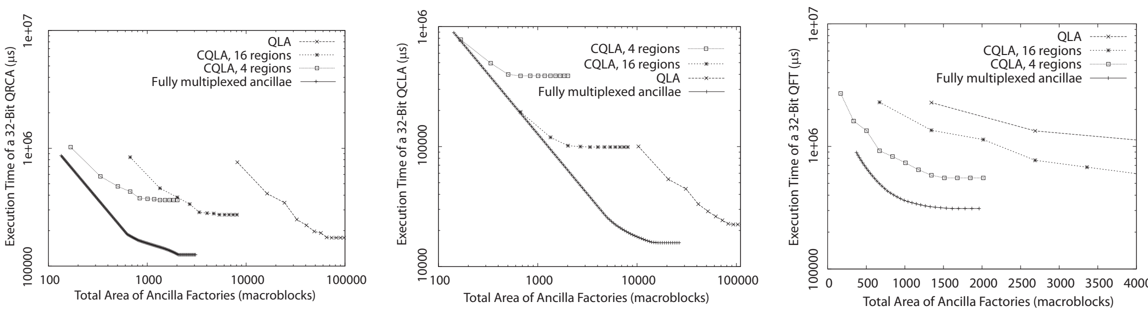


Figure 3.15: Execution time as a function of total area of encoded ancilla factories. (Left) 32-bit QRCA, Data qubit area = 679 macroblocks; (Middle) 32-bit QCLA, Data qubit area = 861 macroblocks; (Right) 32-bit QFT, Data qubit area = 224 macroblocks.

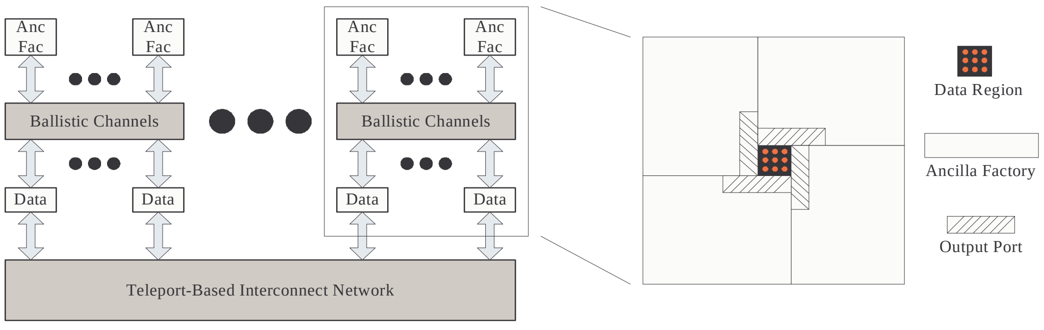
correction after each encoded gate. We simulate dataflow execution taking into account latency of the ancilla production and encoded gate execution, using latencies from Tables 3.5 and 3.8.

CQLA [19] optimizes the QLA design by adding a central compute region of data qubits that are in the current working set. To simulate this specialization of resources, we added tracking of which qubits are in the compute region and account for data qubit fetch and expunge latencies. This simulation has an implementation similar to that of *sim-cache* in SimpleScalar [9]. We used the same basic ancilla production and data gate latencies as for QLA.

Results: Figure 3.15 shows overall circuit execution time as a function of total area dedicated to ancilla factories (of both types) for the different microarchitectures being tested for QRCA (left), QCLA (middle) and QFT (right). Total data qubit area is given in the caption for each.

We notice that CQLA takes about half an order to an order of magnitude longer to execute than Fully-Multiplexed Ancilla Distribution. The chief cause of this is the incurrence of cache misses in CQLA, whereas Fully-Multiplexed always distributes encoded ancillae to data when necessary. CQLA also plateaus half an order to an order of magnitude higher than Fully-Multiplexed since, even with very fast encoded ancilla production, cached misses are still incurred to bring ancillae to data.

QLA requires two orders of magnitude more area for ancilla production to match execution time with Fully-Multiplexed, which is logical since many ancilla generators are idle much of the time in QLA when they could be used to feed nearby data need. On the other hand, QLA eventually plateaus at a similar execution time as Fully-Multiplexed, which makes sense since it has no concept of cache misses. QLA simply needs very high encoded ancilla production at each data qubit in order to run at the speed of data.



(a) Qalypso Microarchitecture (b) Single Tile of Qalypso

Figure 3.16: (a) Qalypso: our proposed microarchitecture. (b) A single tile consists of a dense data region surrounded by ancilla factories funneling encoded ancillae as need arises. Ancilla distribution is fully multiplexed within each tile, with factory output ports placed physically close to the data region.

3.4.3 Qalypso: Microarchitectural Implications of Pipelined Ancilla Factories

The simple encoded zero ancilla factory in Figure 3.9 has an area of 90 macroblocks and a throughput of 3.1 encoded ancillae per millisecond. The pipelined encoded zero ancilla factory designed in Section 3.3.2 has an area of 298 macroblocks and a throughput of 10.5 encoded ancillae / ms. These two designs produce virtually the same encoded zero ancilla bandwidth per unit area. Since classically, pipelining improves the bandwidth of a design for a fixed area, this seemingly negates some of the benefits of pipelining[1].

Nonetheless, we conclude that pipelined ancilla factories provide significant benefit in having concentrated input and output "ports." We propose Qalypso, a tiled microarchitecture shown in Figure 3.16a using the tile shown in Figure 3.16b, with ballistic movement being used within a tile and teleportation of data between tiles [33]. The central data region consists of a dense packing of encoded data qubits and channels for local ballistic movement. The ancilla factories each have an output port physically near the data region so encoded ancillae do not have far to travel. This port is beneficial both in reducing aggregate movement error on encoded ancillae and in avoiding congestion problems from having encoded ancillae generated uniformly throughout an ancilla factory. Meanwhile, since the limiting factor on move speed in ion traps is state decoherence rather than control of the electrodes, stateless qubits may be recycled to factory input ports much more quickly, allowing the input ports to be far from the data.

This architecture differs from (C)QLA in two significant respects. First, our data

[1]We do not gain in bandwidth per unit area due to the facts that the technology is inherently synchronous and that individual gate locations are multi-purpose.

regions consist of data alone. In CQLA, the compute regions consist of both data and ancilla generation units, meaning that data are physically quite a bit further apart even within one compute region and generally require teleportation for movement. Even if QEC were performed as part of teleportation [5], we would require twice as many encoded ancillae as a straightforward QEC step. Thus, we suggest that our data regions be made as large as possible to allow data qubits to reach each other using ballistic movement instead of teleportation as much as possible. Though ballistic movement is somewhat error prone, the area of a data region consisting of nothing but encoded data qubits is still quite small, so teleportation is only necessary between data regions.

Second, ancilla factories surrounding a data region in our design are shared by all data qubits within that region. In Figure 3.14a, which represents the (C)QLA microarchitecture, each ancilla generator is dedicated to a single data qubit (location), so imbalances in encoded ancilla need cause some generators to go idle while others cannot meet need. By having a full crossbar between generators and consumers (data qubits), as in Figure 3.14b, fresh ancillae go where they are needed within a single data region. Further, as shown in Section 3.3.4, the consolidation of ancilla generation into factories surrounding data regions produces a more fault tolerant design than the architecture of (C)QLA.

We now have a microarchitectural structure for our datapath, as well as detailed pipelined ancilla factories. We still need a model for our teleportation based interconnect, and we need a means of finalizing a datapath that is tailored to a given circuit. These two points are the topic of the next chapter.

Chapter 4

Datapath Variations

In this Chapter, we examine the specifics of the Qalypso microarchitecture by designing the compute regions and support infrastructure around these compute regions. In the next Chapter, we will see how to divide large quantum circuits among a series of compute regions.

The Qalypso microarchitecture contains three types of subcomponents:

1. Compute Regions, where data qubits reside
2. Ancilla Factories
3. a teleportation-based interconnect network

Ancilla factories were designed and analyzed in Chapter 3, so we now investigate the other two types of components. We first introduce the network model and components that we'll be using for the remainder of this work. Following that, we investigate some Compute Region layout variations, then our method for automatically sizing the support infrastructure. Section 4.4 will summarize our findings.

4.1 Network Model

The three datapath organizations in Figures 1.10 and 1.11 each contain a high-level view of the network as a grid of teleporters. We now elaborate on how the network is modeled in order to fairly account for both congestion in the teleport network and the area consumed by network components.

4.1.1 Network Components

The circuit for quantum teleportation from Figure 1.8 is depicted more accurately spatially in Figure 4.1. It consists of three steps: EPR Pair Generation, EPR Pair Distribution and Data Teleportation.

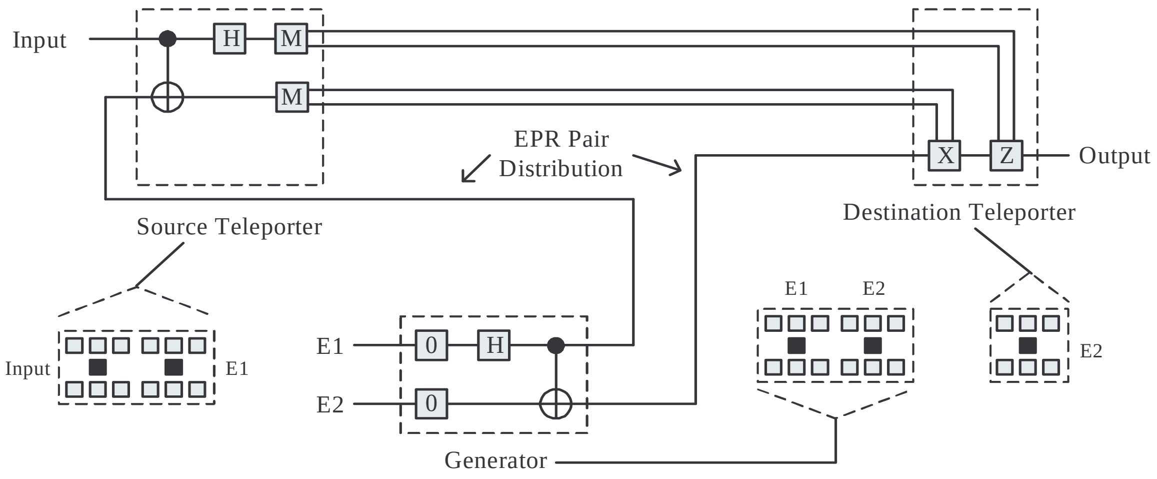


Figure 4.1: The process of teleportation is done in three steps: 1. An EPR pair is created by applying the Generator circuit to two stateless qubits, E1 and E2; 2. E1 and E2 are moved to the source and destination, respectively, of the desired teleportation; 3. The Teleporter circuits are executed, which includes the transfer of two classical bits from source to destination.

EPR Pair Generation An EPR pair is generated by taking as input two qubits, E1 and E2, in unknown state and performing the following steps:

1. Move qubits E1 and E2 into two separate gate locations
2. Prepare both E1 and E2 in the zero state
3. Perform H gate on qubit E1
4. Move qubit E2 into E1's gate location
5. Perform CX gate on qubits E1 and E2

Thus, the hardware for an EPR Pair Generator consists of two Macroblocks (Straight Channels with Gate Location), though channels into and out of the Generator must still be accounted for. The latency for an EPR pair generation is:

$$t_{gen} = t_0 + t_{1q} + t_{2q} = 122\mu s \tag{4.1}$$

EPR Pair Distribution During EPR pair generation, the qubits of the EPR pair must be in proximity to one another. For the data teleportation to occur, one qubit of the pair must be at the source, the other at the target. Thus, during this step, qubit E1 is physically moved from the generator to the source of the desired teleportation, while qubit E2 is moved from the generator to the target. No gate locations are needed during this step, but some means for qubit movement is required, such as ballistic channels.

Data Teleportation The procedure to teleport the data qubit from the *Input* at the source to the *Output* at the destination is accomplished as follows:

1. Perform the CX and H gates at the source, similar to how they were performed (in reverse order) for the Generation step above
2. Measure both qubits at the source, collapsing their state and getting two classical bit values as a result
3. Transfer the classical bit values from source to target using classical wires
4. Perform a corrective Pauli gate on qubit E2 to set its state at the *Output* to match the state that used to be on the *Input* qubit

The source location requires two Macroblock gate locations, much like the Generator does, while the destination location requires only a single one for the corrective Pauli gate. The latency for this compound operation is:

$$t_{tprt} = t_{2q} + t_{1q} + t_{meas} + t_{1q} = 122\mu s \tag{4.2}$$

Note that an X gate followed by a Z gate on the same qubit is the equivalent of a Y gate, so we need only account for the latency of a single Pauli gate on qubit E2.

4.1.2 Purification

As described in the previous section, a Teleportation channel is constructed from a single EPR pair by sending half of the pair to the source of the communication and the other half to the destination. Unfortunately, this prescription ignores the errors accumulated by the EPR pair during transportation. In this section, we introduce “purification,” a process for producing a single noise-free EPR pair at source and destination of a channel by combining a number of noisy EPR pairs. In order to understand the need for purification, we first define the concept of qubit *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 [44]. 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.

We characterize errors by calculating the fidelity of qubits traversing the various quantum channels and gates necessary to route and move bits around a communication network. We will combine models of the individual communication components so that we get an overall *fidelity of communication* as a function of distance and architecture.

In ballistic movement, the fidelity of a bit after going through the ballistic channel over D cells is:

$$F_{new} = F_{old}(1 - p_{mv})^D \tag{4.3}$$

since each hop introduces a probability of error. The time to perform ballistic movement is given in time per cell moved through:

$$t_{ballistic} = t_{mv} \times D \tag{4.4}$$

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}) [18]:

$$\begin{aligned} F_{new} &= \frac{1}{4}\left(1 + 3(1 - p_{1q})(1 - p_{2q})\frac{(4(1 - p_{ms})^2 - 1)}{3}\right. \\ &\times \left.\frac{(4F_{old} - 1)(4F_{EPR} - 1)}{9}\right) \end{aligned} \tag{4.5}$$

The fidelity after a teleportation involves the fidelity of the data before teleportation (F_{old}) and the fidelity of the EPR pair used to perform the teleportation (F_{EPR}).

As shown by Equation 4.5, the fidelity of the EPR pairs utilized in teleportation (F_{EPR}) has a direct impact on the fidelity of information transmitted through the teleportation 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 can be utilized 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. 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 half of the remaining pairs, resource usage is exponential in the number of rounds.

We could implement tree purification naively at each possible endpoint by including one hardware purifier for each node in the tree. However, as the tree depth increases, the hardware needs quickly become prohibitive. Additionally, this mechanism provides no natural means of recovering from a failed purification (loss of a subtree).

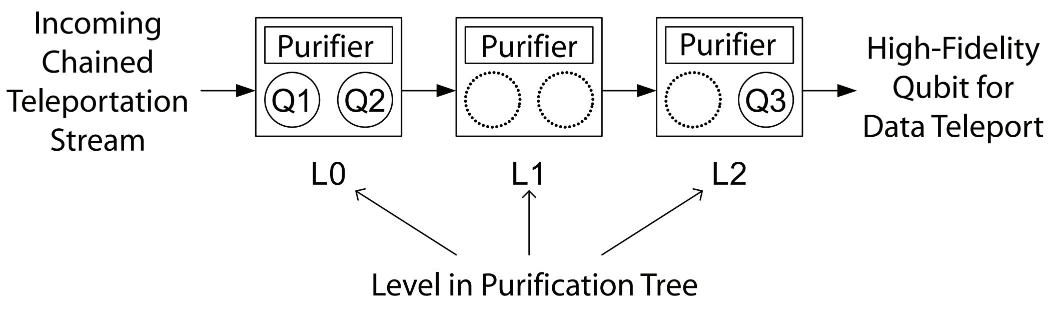

Incoming Chained Teleportation Stream
Purifier
Q1
Q2
Purifier
Purifier
Q3
High-Fidelity Qubit for Data Teleport
L0
L1
L2
Level in Purification Tree

Figure 4.2: Robust tree-based purification: Incoming qubits are purified once at L0, representing the lowest level of the purification tree. Successfully purified qubits are moved on to L1 and purified there, representing the second lowest level, and so on.

A more robust queue-based purifier implementation is shown in Figure 4.2. 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 in a naive implementation). Second, movement between levels of purification is minimized, lessening the impact of movement. Third, no special handling for lost subtrees due to failed purifications 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. For these reasons, we use Queue Purifiers in our simulations.

4.1.3 Teleportation Channel

The problem of communicating quantum information across a large layout can be viewed (somewhat simplistically) as a matter of distributing EPR pairs to the endpoints of each desired communication, followed by performance of the teleportation operations. In this section, we examine the process of EPR distribution in greater detail.

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 4.3. 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.

By performing this process of generation and purification continuously, we create a clean teleportation link between two endpoints. We next describe how to use these links to build a large and manageable teleportation network.

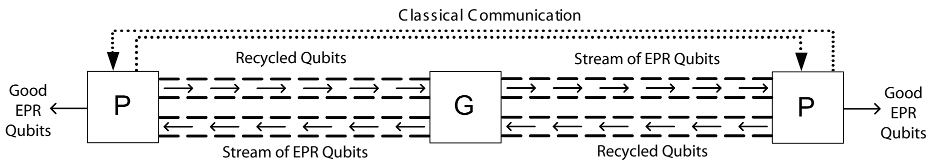


Figure 4.3: Ballistic Movement Distribution Methodology: EPR pairs are generated in the middle and ballistically moved using electrodes. After purification, high-fidelity EPR qubits are moved to the logical qubits, used, and then recycled into new EPR pairs.

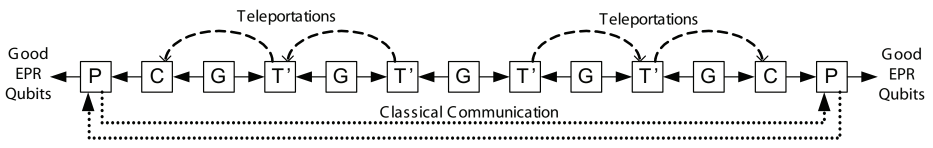


Figure 4.4: Chain Teleportation Distribution Methodology: EPR qubits generated at the midpoint generator are successively teleported until they reach the endpoint teleporter nodes before being ballistically moved to corrector nodes and then purifier nodes.

4.1.4 Chain Teleportation

Another option for distributing EPR pairs over long distances 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 4.4. A T' node contains units that perform the local operations to entangle qubits but no correction capability. 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 predistributed, 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 neighboring 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.

To permit general computation, any functional unit must have a nearby T' node (although they may be shared), necessitating a grid of T' nodes across the chip, which are linked as described above by virtual wires. The exact topology is an implementation choice; one need not link physically adjacent or even nearby T' nodes, so long as enough channels are included to allow each G node to be continuously linking 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.

There are two primary benefits for using Chain Teleportation over purely ballistic distribution of EPR Pairs. First, by performing purification as part of link setup (Figure 4.3), we are achieving a form of *link amplification* that is off the critical path of execution. Second, this structure will facilitates network sizing for our automated datapath designs, since the G and P nodes may be sized automatically to accommodate the T' nodes.

4.1.5 Structuring Global Communication

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.

We assume a mesh grid of routers as a reasonable first-cut at a general purpose routing network [1, 14]. Under the Ballistic Movement Distribution Methodology (Figure 4.3), a routing channel is a straight sequence of ion traps, while a router is at the intersection. Under the Chained Teleportation Distribution Methodology (Figure 4.4), a router is a T' node, and a routing channel is the pre-generated link between two T' nodes. In either case, there must be G nodes distributed across the chip to generate EPR pairs.

Route Planning High-level classical control views the quantum datapath at the logical level. It tracks the current location of each logical qubit but knows nothing of the actual encoding used (*i.e.* number of physical qubits per logical qubit). This control takes the sequence of logical operations that comprise the program and identifies all logical communications that need to occur. It then begins routing them while maintaining program order.

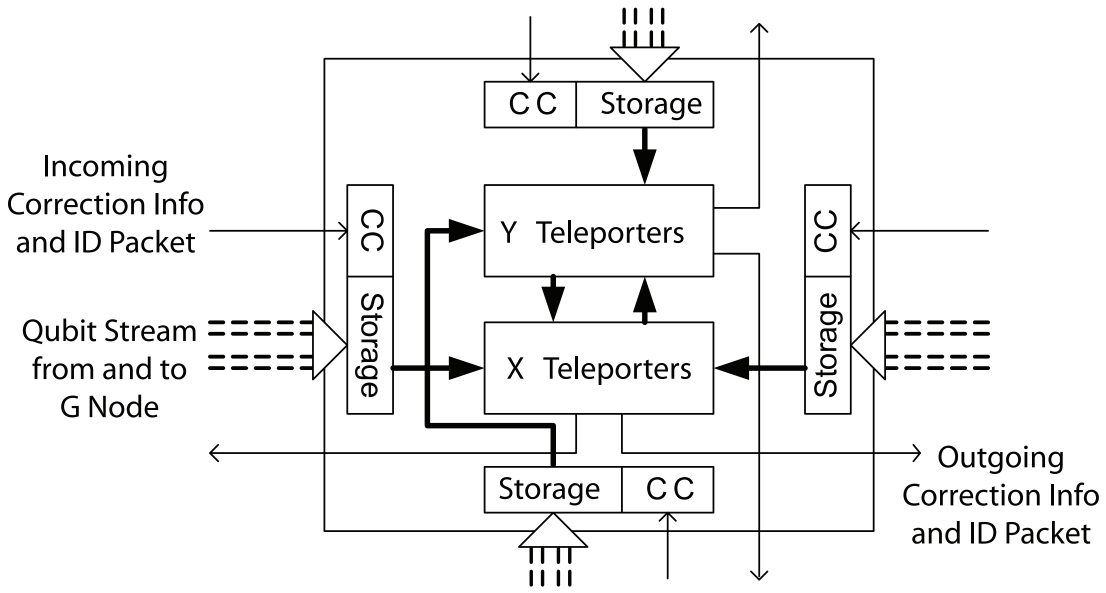

C C
Storage
Incoming Correction Info and ID Packet
CC
Storage
Y Teleporters
X Teleporters
CC
Storage
Qubit Stream from and to G Node
Storage
C C
Outgoing Correction Info and ID Packet

Figure 4.5: A Quantum Router: Two sets of teleporters implement dimension-order routing. Fat arrows are incoming qubits from a G node (and recycled ions in opposite direction). Bold arrows are ion movement within router. Thin arrows are classical data. CC is the classical control including cumulative correction information and further routing.

Once a path has been determined, EPR pairs need to be generated and routed to the endpoints for purification. A G node near the middle of the path is given instructions by the high-level control to generate and name EPR pairs. These EPR qubits are then sent from router to router (whether intersection or T' node) along the routing channels (whether ion traps or teleportation links) until the endpoints are reached, at which point they are locally routed to the purifiers. Thus, under either methodology, the routers need only be able to make local routing decisions based on a qubit's destination.

Local Routing Control Each router and G node needs local classical control to determine how it handles qubits, which requires a means of identifying qubits. Thus, each qubit is associated with a classical message which travels alongside the qubit in a parallel classical network. The node control for the G node which generates a pair also generates their accompanying messages. 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 information that will be used at the endpoint. A router forwards a qubit on to the appropriate routing channel or to a local corrector at the destination.

Figure 4.5 shows one possible implementation for a router. The router receives a constant stream of EPR pairs (from G nodes) linking it to its neighbor routers. During an incoming teleportation, a qubit enters the Storage area to wait for the operations at the teleportation source to complete. Classical data in the form of the teleporting qubit's ID packet and the two classical bits used in the teleportation enter the adjoining classical control (CC). The cumulative correction information is updated in the ID packet, and the destination information is used to route the qubit to the correct set of teleporters (or to a purifier if an endpoint has been reached). For an outgoing teleportation, a qubit from the G node stream bypasses the Storage area and moves directly to the appropriate teleporter.

In this router design, the teleporters are divided into two sets. One set handles all traffic moving in the X direction, the other handles traffic moving in the Y direction. For a turn, an EPR qubit must be ballistically moved between the teleporter sets (as shown by the bold-headed arrows). It is evident from the crossing arrows in the figure that streams of qubits may need to cross. However, even with stalling, movement time is so much faster than teleportation (Table 2.2) that crossing will not be a limiting factor.

4.2 Compute Region Layout

In this section, we are going to switch gears and examine the structure of the compute regions. We will have to address both the internal structure of compute regions as well as the interaction between compute regions and the rest of the system (both ancilla generators and networking). We start with the internal structure.

4.2.1 Designing the Interior of a Compute Region

Figure 4.6 shows three styles of layout for Compute Regions. Figure 4.6a contains a vertical channel for communication by ballistic movement. Each qubit can reach the location of exactly one other qubit without performing any turns and can reach any of the other qubits in the Region with exactly two turns. Note that these are *logical* qubits, so if we're using the Steane [[7,1,3]] code, each represents seven physical qubits and the layout is actually seven times taller than it appears. Variations on this design include changing the number of rows and changing the number of vertical channels to alleviate congestion problems, but the number of columns of qubits is fixed at two.

If we want to have more than two columns of qubits in a single Compute Region, we need horizontal channels for movement within the Region. Figure 4.6b shows such a design. Exterior channels increase the size of the Region but do not push the qubits within the Region further apart from one another. Interior channels provide a more direct route for movement, but they separate the qubits. Variables in this design include both vertical and horizontal channels, as well as qubit count in both

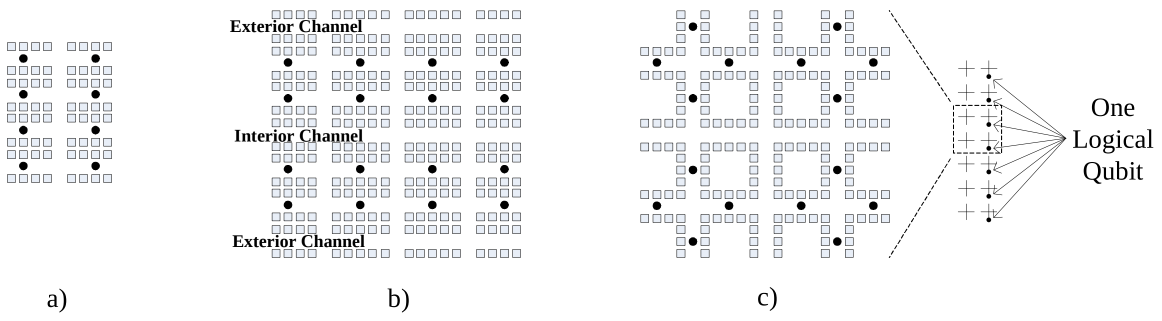


Figure 4.6: Three Compute Region layouts: a) Two columns of logical qubits. Data qubits use the central pathway; connections to ancillae and network are all around the perimeter. b) With more than two columns, data qubits need horizontal channels for communication within the Region, with either interior or exterior horizontal channels. c) Cross-based layout to give groups of qubits more options to avoid turns. Note that the physical qubits of a single logical qubit must be separated, so this figure depicts a *physical* layout.

directions.

Something that both these designs have in common is that qubits are always an even number of turns apart. Figure 4.6c shows a layout in which each qubit can reach two other qubits with zero turns, two more qubits with one turn each, and the rest with two or more turns. The catch is that since data qubits interact transversally, the corresponding *physical* qubits of the two interacting logical qubits must be next to each other. Thus, Figure 4.6c shows a physical level layout, so the physical qubits of each logical qubit are spaced apart. The result is slightly more complicated distribution paths for encoded ancilla qubits and encoded EPR pair sets.

4.2.2 Interfacing with the Network

Having designed the Compute Regions, we now examine how they interface with ancilla generation and the network. The specifics may be varied since Qalypso is a malleable microarchitectural specification, but Figure 4.7 gives a good idea of the extent of the design space.

The Compute Region (CR) is sized to accommodate the desired number of logical data qubits. The number of available interface ports (black rectangles in the figure) are determined by the perimeter of the Compute Region. The components of the support infrastructure are allocated ports as necessary. Note that, depending on the characteristics of the circuit, not all ports need be used. In Figure 4.7, the top two ports are not used by any of the support infrastructure.

In the case of ancilla factories, the output ports are connected to the Compute Region. In the case of the teleport network, the output of each Queue Purifiers must

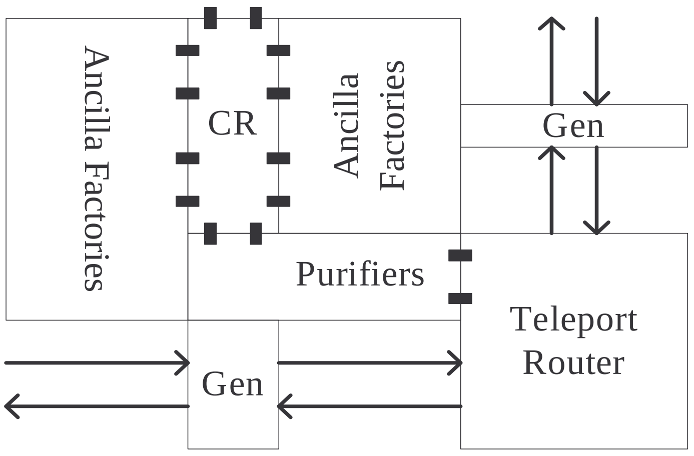


Figure 4.7: The Compute Region (CR) is surrounded by the various components of the support infrastructure. The black rectangles are interface ports for encoded qubits to move ballistically between datapath regions.

be attached to a Compute Region port. Since a great deal of congestion can occur at Teleport Routers as links are being established, we size the Generators and Queue Purifiers associated with each Router to be capable of sustaining maximum possible bandwidth. Thus, the problem of sizing the teleport network is reduced to simply sizing the Teleport Routers.

Thus, a finalized Qalypso datapath requires that the following be specified:

1. Compute Region size and structure
2. Ancilla Factory counts and port allocations
3. Teleport Router sizing (and accompanying sizing for Generators and Queue Purifiers)
4. Queue Purifiers port allocations

With our design space clearly defined, we may now explore the problem of determining an optimal datapath for a target circuit. We handle Compute Region layout first, followed by support infrastructure sizing.

4.2.3 Choosing Between Layouts

To choose between possible compute region organizations, we map a large random circuit and examine ADCR as a function of organization. We first investigate variations of the layout in Figure 4.6b. Figure 4.8 shows the results of running a 256 qubit, 1 million gate circuit on various such layouts. Note that the plotted points

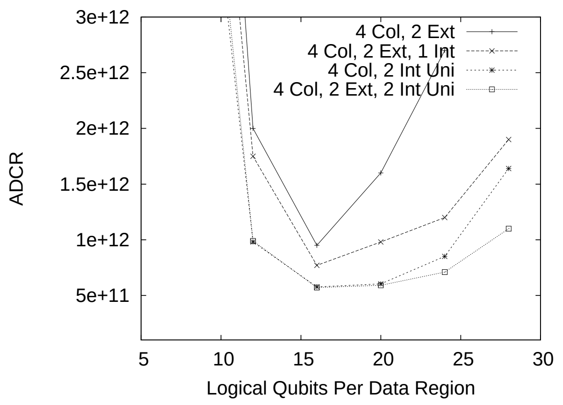


Figure 4.8: Random graphs of 1 million gates on 256 qubits run on varying versions of the Compute Region in Figure 4.6b. Having two unidirectional Interior Channels is key, while Exterior Channels help a little bit. (Col = Number of Columns of Qubits, Ext = Exterior Channel, Int = Interior Channel, Int Uni = Unidirectional Interior Channel)

are ADCR-optimal, meaning that the network and ancilla factories are sized as will be discussed in Section 4.3. Our only variable in these tests is the internal Compute Region structure.

From the data in Figure 4.8, we see that two matters are key. First, Unidirectional Interior Channels make a big difference, largely due to the fact that local routing becomes more organized. When ancilla qubits and encoded EPR pairs are included along with data qubits, there are simply an immense number of qubits moving around. Qubits taking arbitrary shortest paths along Bidirectional Interior Channels result in more erratic communication patterns and more stalls. Second, while the 2 Ext, 2 Int Uni layout does provide the optimal point by a small margin, the data points on the plot are averaged over the runs of 1000s of random graphs, meaning that we cannot declare a single optimal layout for all graphs. The solution is to have the mapper actually try different layouts and pick the optimal one for the target circuit.

We next investigate variations of the layout in Figure 4.6c. Figure 4.9 shows the results of running a 256 qubit, 1 million gate circuit on various such layouts. The optimal point occurs at 12 qubits per Compute Region, as opposed to 16 in Figure 4.8. The reason is that encoded qubits in this layout style do not always move transitively, making stalls from collision avoidance occur more frequently in larger Compute Regions.

Figure 4.10 compares the best versions of each of the three designs in Figure 4.6 for random graphs of 1 million gates on 256 qubits. Option a is a restricted version of Option b, so the flexibility of b wins out. Option c suffers from the difficulty of effective automated pathfinding on a physical qubit level. Thus, given adequate time and resources, the most thorough mapper would search over variations of Option b

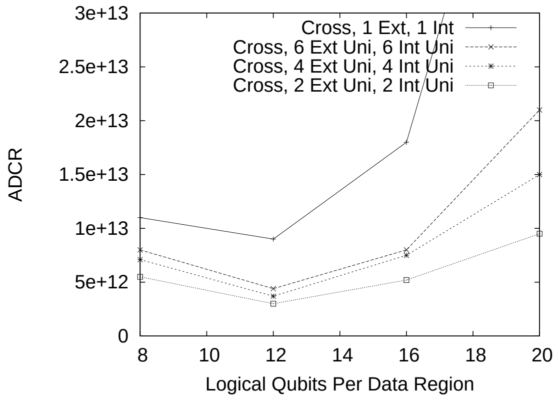


Figure 4.9: Random graphs of 1 million gates on 256 qubits run on varying versions of the Compute Region in Figure 4.6c. (Cross = Layout in Figure 4.6c, Ext Uni = Unidirectional Exterior Channel, Int Uni = Unidirectional Interior Channel)

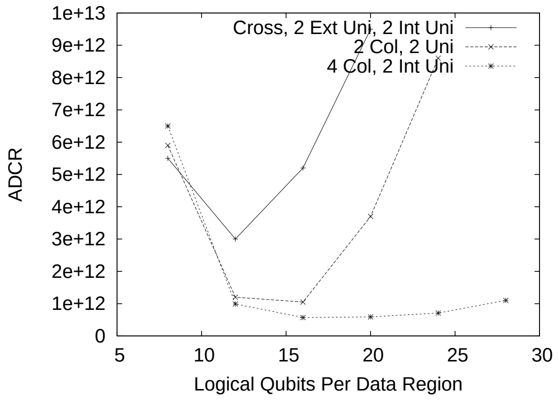


Figure 4.10: Random graphs of 1 million gates on 256 qubits run on the layout in Figure 4.6a and the winners from Figures 4.8 and 4.9. The layout design in Figure 4.6b results in the best ADCR.

to find the best Compute Region layout for a given circuit.

4.3 Sizing Ancilla Factories and the Teleport Network

A first approach to sizing the support infrastructure might be to initialize it to zero area, then to allow resizing as necessary during a single mapping pass through the graph. This online resizing is ill advised for two reasons. First, it would likely result in imbalanced node sizes in the final datapath, which results in empty wasted

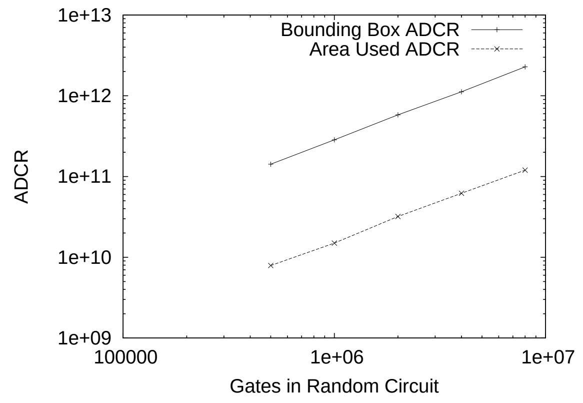


Figure 4.11: If we choose a Compute Region layout, then map the circuit, sizing each node's support infrastructure exactly as necessary without trying to balance area, we end up cheating. The result is that the reported used area ends up being 20x smaller than the bounding box area due to wasted space.

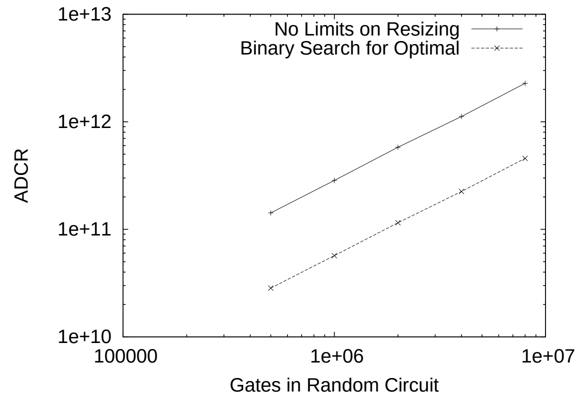


Figure 4.12: By fixing support infrastructure size on each mapping pass and performing a search for optimal sizing, we achieve a 5x improvement in ADCR. This gain is due to the fact that a balanced support infrastructure significantly reduces wasted space on the datapath.

space within the datapath. Figure 4.11 shows that the useful area of the datapath is a poor metric of success, as wasted space could cause the actual bounding box area of the datapath to be more than an order of magnitude larger.

Second, resizing ancilla factories and the teleport network in the middle of a single graph traversal often results in the weakening of earlier decisions. Resizing downwards may result in previous choices having insufficient resources, while resizing upwards may make previous choices less optimal.

The solution is to place a hard limit on support infrastructure size per node on each mapping pass and perform a search to determine optimal sizing. In this way, we get the benefit that decisions made throughout each mapping pass remain valid since the datapath is not changing, but we search over multiple datapath configurations to find the best. Figure 4.12 shows that we gain significant benefit over the naive approach of resizing during a single mapping pass.

4.4 Summary and Findings

In this chapter, we have investigated several key pieces of infrastructure. We have concluded that while no single optimal compute region necessarily exists for all circuits, the key to avoiding congestion problems within compute regions is to provide sufficient unidirectional channels. The latency gain from the smoother ballistic movement overwhelms the increased area required by the channels.

Further, we have concluded that the sizing of the support infrastructure, both ancilla generation units and the teleport network, is best done by doing complete ADCR evaluation on each prospective datapath. That is, rather than attempting to adjust support resources while mapping the quantum circuit, we should make multiple mapping passes, each time on a different fixed datapath, until we find the best sizing for each datapath component.

For the remainder of this thesis, we will use these techniques. It remains for us to decide how best to partition large quantum circuits across a series of compute regions, while at the same time sizing the ancilla regions, network components and compute regions. That is the topic of Chapter 5.

Chapter 5

Mapping Quantum Circuits to Qalypso

The goal in mapping a quantum circuit, as shown in Figure 5.1, is to produce a finalized datapath and associated mapping onto that datapath, both of which are optimized for the target circuit. The circuit corresponds to a dataflow graph, with each vertex representing a one or two qubit logical gate. The datapath consists of sets of multi-purpose logical gate locations in Compute Regions, which are supported by ancilla factories and a teleport network. The datapath specification may also include pre-designed components, such as ancilla factories (Chapter 3) and queue purifiers (Section 4.1.2).

A finalized datapath refers to a realizable layout with sizing and placement for gate locations, channels for ballistic movement and all necessary support infrastructure. The mapping specifies for each dataflow graph vertex the logical gate location where that vertex (gate) will be executed. It does not specify exact timing, as timing may be affected by unpredictable online events such as purification failures, however simulation of the mapping (including errors) can provide more exact timing and overall success probability. Combined with the calculated area of the final datapath, we may evaluate the mapping using the ADCR metric introduced in Section 1.3.3.

In addition to desiring a low latency and high resource utilization, which are standard goals for mappers, the complexity in this problem arises from two sources. First, the support infrastructure dwarfs useful computation in terms of resource needs. Both encoded ancilla preparation and EPR pair creation and distribution require significant advance work to be done, and each may stall program execution if resources are insufficient. Likewise, the interaction of data with ancillae and EPR pairs requires a measurement operation (which is on the critical path), for which the latency is several times the latency of a useful two qubit gate. Thus, a poorly mapped gate could result in penalties far greater than the cost of the gate itself.

Second, the datapath is malleable and may be tailored to fit the circuit. This flexibility is beneficial, however it also expands the search space considerably. The

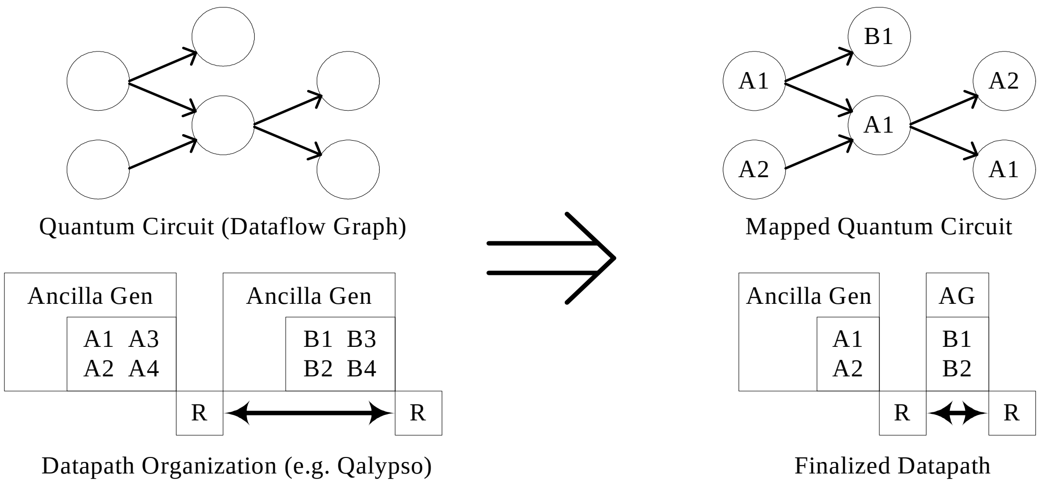


Figure 5.1: The goal of the mapper is to map each operation in the quantum program to a physical resource on the datapath while simultaneously finalizing the datapath configuration, if necessary.

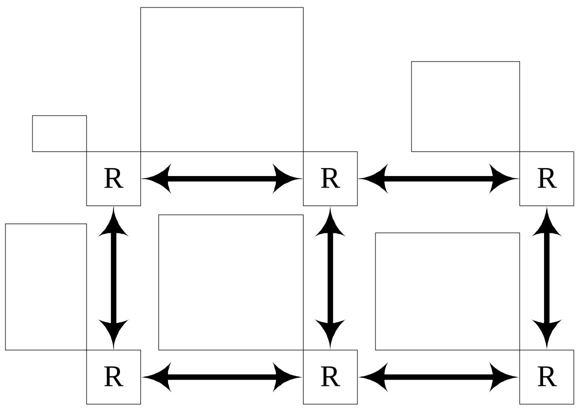


Figure 5.2: The bounding box area of this imbalanced datapath is drastically different from its useful area. An accurate evaluation of ADCR using a datapath should use the bounding box area, which penalizes wasted space.

final datapath may be adjusted for the target circuit, but it must also be realizable. The datapath in Figure 5.2 is imbalanced, resulting in different values for bounding box area and useful area. In Figure 4.11, we showed that blind evaluation of ADCR using useful area is a significant cheat.

Prior work in the quantum realm has involved analogues of fixed classical datapaths. QLA is akin to a quantum FPGA, while CQLA adds the concept of a memory hierarchy to the design. Both are logical but fail to exploit the possibilities inherent in a malleable datapath, which is why we use the Qalypso framework for the mapping step. The mapping heuristic used by CQLA is a basic greedy heuristic, which is the baseline approach upon which we improve in this work.

5.1 Overview of Mapping Techniques

We investigate two primary types of mapping techniques. We first adapt classical priority list scheduling to the problem of mapping a quantum circuit, using a greedy approach to gate allocation. Following that, we introduce the more advanced technique of defining qubit *Homes*, which allow us to better control localization of operations involving critical qubits.

List Scheduling For our baseline, we use classical priority list scheduling to map the quantum circuit onto our datapath. The two primary variables which need to be defined are as follows:

- Vertex Priorities: These determine the order in which logical gate operations are mapped.
- Objective Function: This is the function used to determine optimal mapping location for each logical gate operation.

Both of these will be investigated in Section 5.2.

Home-Based Mapping A more advanced technique that we will examine in this chapter is the use of qubit *Homes*. Each qubit is assigned a Home where it performs its operations and to where it moves if it is expunged from another location. Qubits are ranked by criticality (total gate count). In the case of two qubit operations, the gate is performed at the Home of the more critical qubit.

There are a few reasons why Homes may be beneficial. First, the most critical qubits will have other qubits brought to them. While network setup may be done without stalling data, the actual teleport operation is on the critical path of the data qubit. In order to execute the second CX gate in Figure 5.3a, qubits Q0 and Q2 must be brought together. Since Q0 is involved in the first gate, teleporting Q0 to Q2 must be done after completion of the first gate (Figure 5.3c) while teleporting Q2 to Q0

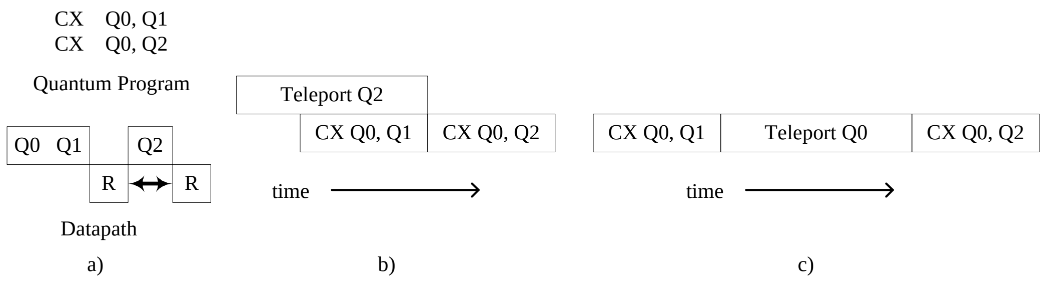


Figure 5.3: We wish to execute the two gates in a) on the datapath below them, with qubits Q0 and Q1 in one Compute Region and Q2 in another. Teleporting Q0 to Q2 must be done serially with the first gate (b), while teleporting Q2 to Q0 may be done in parallel. Times are not to scale.

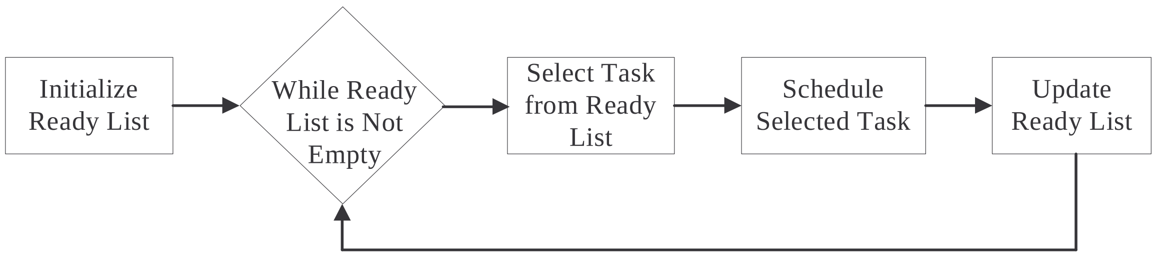


Figure 5.4: List scheduling involves keeping a priority-order list of unscheduled tasks whose dependencies have been fulfilled. One by one, the tasks are polled from the list and scheduled. After each step, the list is updated to account for any newly fulfilled dependency.

may be done in parallel (Figure 5.3b). Thus, the most critical qubit will sit in place and have a stream of other qubits brought to it, decreasing the latency of the critical path and thus execution latency.

Second, well-assigned Homes insure that there is room for critical qubits in the middle of the datapath. Without Homes, a few unfortunate non-critical gate mappings could result in clogged central Compute Regions, pushing critical qubits far apart. Third, Home assignment can be used to hold together highly interactive sets of qubits, which we call *clicks*. If these can be identified by graph analysis, they can be positioned together rather than relying on the hope that greedy gate mapping will keep them in close proximity.

5.2 Priority List Scheduling

The most efficient heuristic algorithms for scheduling are based on list scheduling [27]. The basic structure of a list scheduling algorithm is shown in Figure 5.4.

The *Ready List* contains the set of tasks available to be scheduled at the moment, which is restricted by the dependencies in the task dependency graph.

The Ready List is initialized to contain all tasks which have no input dependencies, i.e. the source nodes of the graph. As long as any task remains unscheduled, the highest priority task in the Ready List is scheduled. Then, the Ready List is updated by removing the newly scheduled tasks and adding to the List any tasks whose dependencies have now been fulfilled.

List schedulers differ in two key components:

1. Task priorities: While the Ready List is not empty, the highest priority task is removed to be scheduled. The prioritization of tasks must be clearly defined and likely impacts the final scheduling.

2. Objective function for scheduling a task: The process of scheduling a task requires specification of the hardware onto which we're scheduling and an objective function for selecting the optimal resource for task assignment.

In the remainder of this section, we introduce and evaluate a few basic approaches to these two components for the problem of scheduling a quantum circuit onto our Qalypso architecture.

5.2.1 Vertex Priorities

In Figure 5.4, the "Select a Task from Ready List" step sets vertex processing order, which determines prioritization of vertices as they are mapped onto the datapath. We now investigate prioritization of vertices by three criteria: qubit criticality, dataflow order and critical paths. Below are summarized the most successful prioritizings involving these criteria.

Greedy By Qubit: Qubit criticality is equal to the qubit's total gate count over the full run of the program. Sort all data qubits from most critical to least. Map all gates for the most critical qubit first (in dataflow order). Next, map the next most critical qubit, and so on until all vertices have been mapped.

Greedy By Centered Start Times: Set the end time for each sink vertex to time 0. Perform a reverse traversal of the graph. Schedule each vertex to occur as late as possible, ignoring both communication latency and ancilla needs (all times will be ≤ 0). Upon completion, keep only the start times for source vertices. Add a constant to each source vertex start time such that the earliest source vertex starts at time 0 (while the rest start at a time ≥ 0). Perform a forward traversal of the graph using these start time. Schedule each vertex to occur as early as possible, again ignoring communication and ancillae. This process serves to *center* the graph. In the absence of communication and ancillae, these are the optimal start times for all vertices such

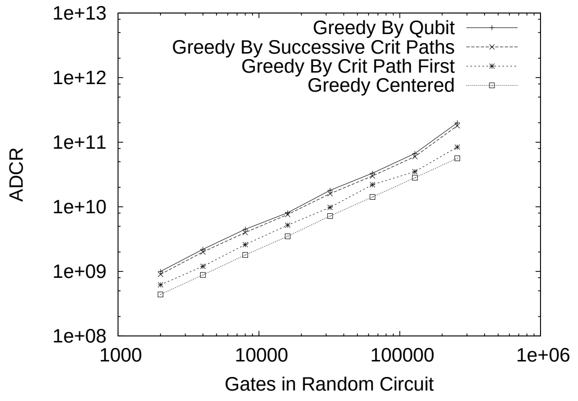


Figure 5.5: Vertex prioritization order based on qubit criticality or critical paths suffer from the drawback of not having full system state information available when each vertex is mapped. The Centered Start Times approach makes decisions based on complete knowledge of the utilization of the support infrastructure, which makes it the most successful.

that no stalls occur and qubit are in existence for the shortest time possible (which is beneficial due to idle errors).

Greedy By Critical Path First: Find a critical path through the graph. Map this critical path in dataflow order. Then use Greedy By Centered Start Times on the rest of the graph.

Greedy By Successive Critical Paths: Find a critical path through the graph. Map this critical path in dataflow order. Remove all vertices on this path from the graph. Find a critical path through this (possibly disconnected graph). Map this critical path in dataflow order, then remove all these vertices. Repeat until all vertices have been mapped.

Figure 5.5 shows the effectiveness of the vertex prioritization metrics described above. Greedy By Qubit suffers from two drawbacks. First, by handling vertices by qubit, it consistently maps over the full program run on each step, not getting a full picture of the system state at any given moment until the end, meaning that utilization of the support infrastructure is short-sighted and often quite imbalanced. Second, in both random graphs and in the adders, many qubits vie for nearly equal criticality. This approach might have benefits if qubit criticality was truly distributed, but none of the benchmarks exhibit this behavior, nor do any proposed quantum algorithms in the literature.

Greedy By Successive Critical Paths likewise suffers from the drawback of making mapping decisions on insufficient knowledge of system state. Greedy By Critical Path First performs better, primarily due to the superiority of Greedy By Centered

Start Times. By mapping only a single critical path, it localizes all the gates along that path, which may not be a good idea due to a corresponding heavy ancilla need. However, the rest of the vertices are then mapped with better knowledge of system state.

Greedy by Centered Start Times maps the vertices in what would be the ideal ordering in the absence of communication and ancillae. While the true ordering may vary slightly due to congestion and limited hardware, the estimate of system resource utilization at the time of vertex mapping is highest in this case, which gives the mapper the best shot at making a good decision.

5.2.2 Options for "Best" Gate Location

The "Schedule Selected Task" box in Figure 5.4 determines the location on the datapath to which a vertex corresponding to a two qubit gate will be mapped given current knowledge about system state, including the location(s) of any involved qubits. (Single qubit gates and QEC steps are simply performed at the qubit's current location.) We investigate some options for quantifying the value of "best."

Four factors are key to selecting a gate location for a vertex V. First, if a location currently contains a qubit not involved in V, then that qubit would have to be expunged from that location in order to perform V there, which involves extra communication. Thus, empty gate locations receive preference.

Second, based on the gates currently being executed, we can determine qubit criticality. Since we have the complete dataflow graph available to us, we can look into the "future" to estimate which qubits will be needed sooner rather than later. These qubits are given preference in not being expunged.

Third, if the next operation for either qubit will be a QEC step or a non-transversal gate, then the corresponding ancilla will be required. In this case, the demand on ancilla production at that Compute Region must be considered.

Fourth, communication cost is a concern. If either qubit needs to be teleported, then that not only involves considerable work by the network to prepare the teleportation link, but the data interaction required impacts the latency of execution even if the link is successfully prepared in time. Ballistic movement cost is also included, but generally as a tiebreaker since teleport latency and link setup cost are far greater penalties.

We now compare a few different approaches to gate location selection. Note that when determining a gate location for a two qubit gate involving qubits $q0$ and $q1$, a gate location is considered "empty" if it contains no qubits other than $q0$ and $q1$. For example, if $q0$ is alone in a gate location, that location is considered empty for the purposes of scheduling this instruction.

Move Only One Qubit: The designers of (C)QLA suggest always performing a gate at the current location of one of the involved qubits, specifically the source qubit in a two qubit operation. This simple approach involves minimal processing time to make a decision.

Empty Loc "Between" the Qubits: Search over all empty gate locations on the datapath and pick a random one that is within the bounding box created by the current positions of the qubits, if possible. This simple approach attempts to minimize network use but does not take ancilla production into account.

Best Empty Loc When Penalizing SI: Search over all empty gate locations within the bounding box created by the current positions of the qubits, but penalize based on support infrastructure (SI) usage. Given knowledge of current and past resource usage in the network, compute how long the qubits would have to stall for the network to set up the desired link, if at all. (If network resources are available to set up the link in time, there is no penalty.) Further, if the next gate for either qubit required an ancilla, compute how long that gate would be stalled based on ancilla availability, assuming nothing else stalls. Take the maximum of these two as the stall penalty for this gate location.

Best Compute Region When Penalizing SI: Perform the same stall penalty calculations as above, but do so for each Compute Region rather than for each empty gate location, which is possible since each Compute Region has fully multiplexed ancilla factories and a single port to the network. When the best Compute Region is chosen, pick an empty gate location. If none are empty, expunge the least critical qubit from its gate and send it to an empty gate location at the nearest edge of the datapath.

Results: Selecting a random empty gate location between the two qubits involved performs marginally better than always using the location of the source qubit. The reason is that it gives highly active qubits a chance to move together rather than forcing them to adhere to the initial placement. However, both of these suffer from the fact that failing to take the support infrastructure into account results in unexpected stalls and an imbalanced datapath with wasted space.

Significant speedup is achieved by accounting for limited resources in the network and ancilla factories. It also turns out that expunging qubits is not worthwhile since the cost of the extra teleport setup outweighs any gains in gate mapping.

5.2.3 Issues with Naive Mapping

The primary drawback of the approaches shown so far is that there is no consideration for the business of a qubit in the near future. Gates are scheduled largely based on their criticality in the dataflow graph, meaning that when a gate is to be scheduled,

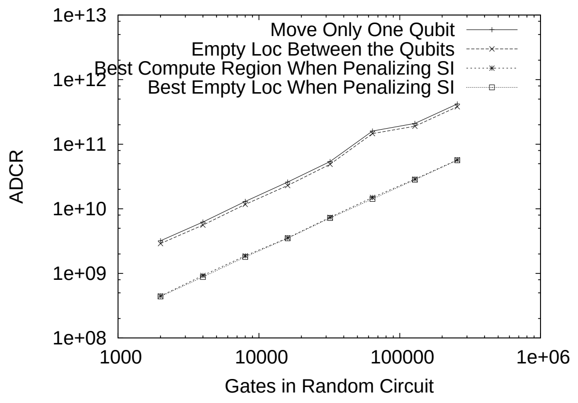


Figure 5.6: A comparison of four different definitions for "best" gate location. The best result occurs when both components of the support infrastructure (SI) are accounted for and when empty gate locations are used (so qubits don't need to be expunged).

Compute Region Size	Local Movement of Critical 50% Set	Inter-CR Movement of Critical 50% Set
4 data qubits	3.1	96.9
8 data qubits	6.8	93.2
16 data qubits	12.7	87.3

Table 5.1: Using the basic list scheduling approach outlined in this chapter, we find that critical qubits (the minimal set that accounts for at least 50% of total gate count) experience very high inter-Compute Region (CR) communication, which results in a larger or heavily delayed teleport-based interconnect.

the relative importance/business of the two qubits involved is not incorporated into the decision process. As a result, critical qubits could end up teleporting arbitrarily and repeatedly across the datapath.

For the moment, let us consider the minimal subset of data qubits which account for at least 50% of gate operations in the graph. Let us call these the Critical 50% Qubits. For random graphs on 100 qubits and 20000 gates, Table 5.1 shows that the amount of inter-Compute Region movement for these qubits is unacceptably high, thus putting extreme strain on the network, which results in ballooned area or latency or both.

Ideally, we wish to avoid having critical qubits teleporting willy-nilly across the full diameter of the datapath due to poor planning for the near future. To this end, we explore in the next section various means to encourage the most critical qubits to remain near the center of the datapath and have less critical qubits brought to them.

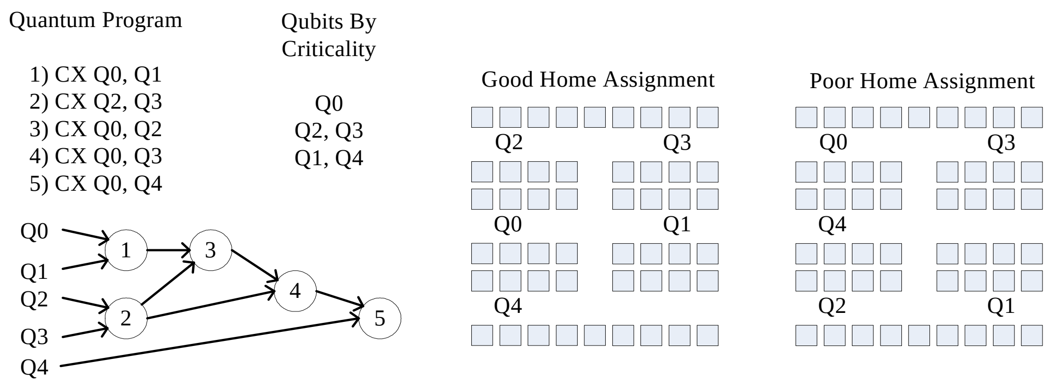


Figure 5.7: This quantum program involves five qubits, Q0 being on the longest critical path, Q1 and Q4 on the shortest. In home-based mapping, each two-qubit gate is performed at the "home" of the more critical qubit involved. Shown are good and poor home assignments for this particular program.

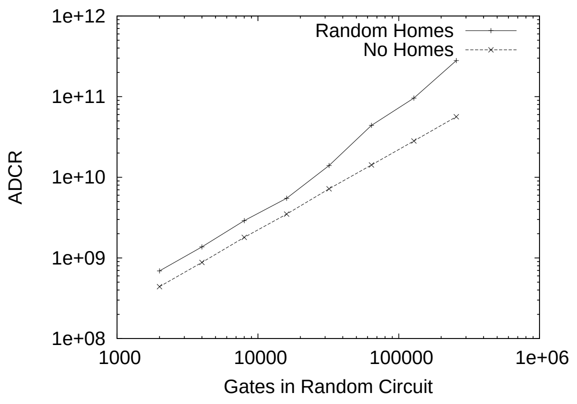


Figure 5.8: Assigning and enforcing random Homes is worse than not assigning them at all. More intelligent means need to be devised.

5.3 Home Assignment By Circuit Analysis

Figure 5.7 shows examples of good and poor Home assignment for a simple circuit. In the good assignment (center of the Figure), the most critical qubit, Q0, is placed centrally and the first two gates may be performed without any turns. In the poor assignment, congestion would stall the run because the first two gates collide with each other and because Q0's Home is less accessible.

5.3.1 Random Home Assignment

Figure 5.8 compares a mapping heuristic which uses randomly assigned homes against the best heuristic from the previous section, which is referred to as *No Homes*. These results show that enforcing Homes without intelligently assigning them has a detrimental effect on the mapping. Thus, we next investigate deterministic methods for assigning qubit Homes.

5.3.2 Critical Qubit-Based Home Assignment

Mapping decisions for critical qubits have a greater chance of impacting the final metrics than those for less critical qubits. For this reason, we investigate the effectiveness of assigning Homes based on qubit criticality. In all cases, qubit criticality is defined by its total gate count (the higher the count, the more critical the qubit).

- **Home Crit Qubit Together:** Assign Homes to qubits in order of descending criticality. Assign Homes for critical qubits as close to each other as possible (incorporating the penalties for teleportation). Thus, the most critical qubits will be assigned Homes in the center of the datapath (the most accessible region), with successively critical qubits assigned Homes progressively outward.

- **Home Crit Qubit Separate:** Assign Homes to qubits in order of descending criticality. Assign successive qubits to different Compute Regions in order to distribute congestion and ancilla needs. Homes are assigned starting in a central Compute Region and moving outward, but each Region is assigned an n^{th} Home before any Region is assigned an $(n+1)^{st}$ Home.

5.3.3 Interaction-Based Home Assignment

Ideally, we want groups of highly interactive qubits grouped together and non-interactive qubit pairs in separate Compute Regions. We now investigate heuristics based on identifying interactive qubit groups, which we refer to as *clicks*. For each of these heuristics, we begin by recording the number of interactions between each pair of qubits in the quantum circuit.

- **Exhaustive Search:** For very small problem sizes, we may exhaustively search all possible groupings of qubits to find the optimal Home assignment. While thorough, this approach has a factorial running time, so it is only practical on small circuits.

- **Home Interaction By Critical Qubit:** Start by assigning the most critical qubit Q_0 to a central Compute Region in the datapath. Fill in the rest of this Compute Region with qubits having the highest interaction count with Q_0.

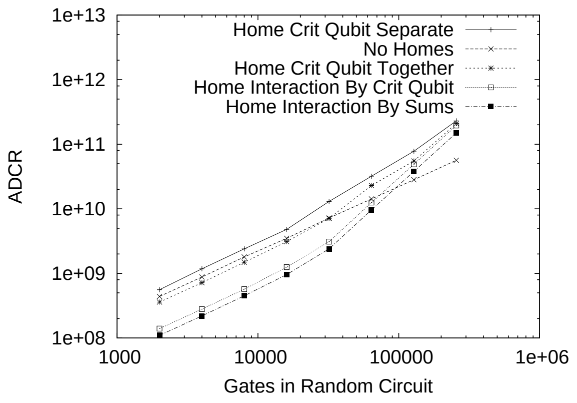


Figure 5.9: Intelligent assignment of Homes does in fact beat the basic Greedy approach, however past a certain circuit size, changing communication patterns over the course of a run makes a single assignment of Homes inadequate.

Repeat this process with unassigned qubits until all qubits have been assigned a Home.

- **Home Interaction By Sums:** Select the two qubits, Q_0 and Q_1, with the highest interaction count. Assign them Homes in the most central Compute Region CR. Next, add the qubit Q_2 with the highest interaction count with all qubits in CR (the sum of Q_2's interaction counts with all qubits already assigned to CR). Repeat until CR is filled. Once CR is filled, repeat this process for successive Compute Regions until all Homes have been assigned.

5.3.4 Comparison Between Deterministic Techniques

Figure 5.9 compares the various Home assignment heuristics introduced in this section as applied to random graphs. The baseline for comparison is the "No Homes" heuristic, refers to the best Greedy heuristic in Figure 5.6. Home Interaction By Sums wins overall and provides a significant improvement in ADCR over No Homes.

However, all Home assignment heuristics start failing for larger graphs due to the fact that the communication clicks change over the course of a longer run, so no single Home assignment can encapsulate the entire graph. This deficiency will be addressed in Section 5.5.

Exhaustive search of Home placements allows us to determine how good these results are. Unfortunately, it's only feasible to exhaustively search small problem spaces. Figure 5.10 shows a small datapath with room for eight qubits. Figure 5.11 compares the Home Interaction By Sums heuristics against the optimal Home assignment (as determined by exhaustive search) on random graphs on eight qubits. Home Interaction By Sums is within 1% of optimal for these problem sizes.

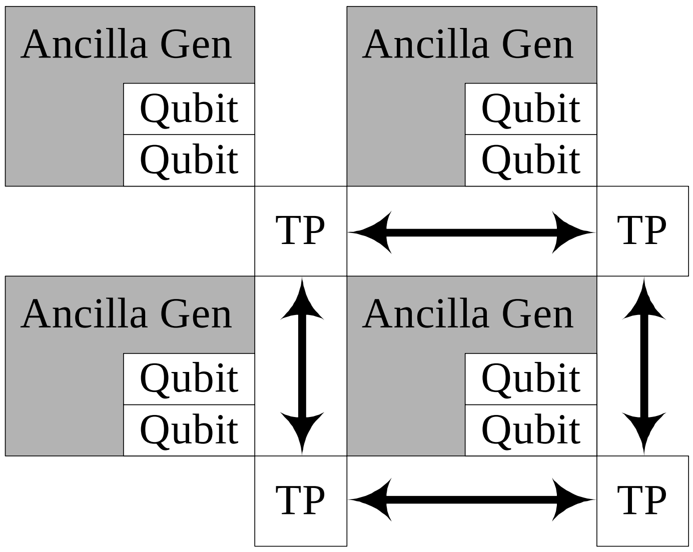


Figure 5.10: This datapath, with room for eight logical data qubits, is used to obtain the exhaustive search results in Figure 5.11. Not to scale.

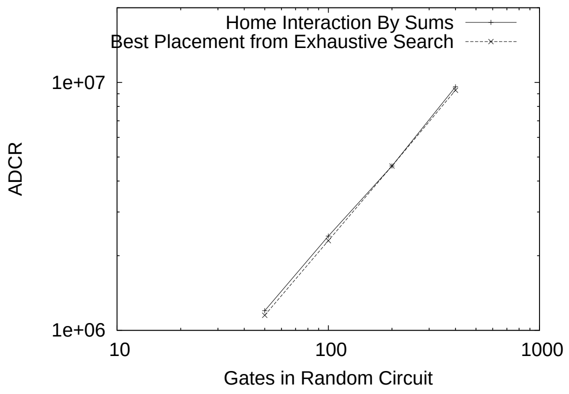


Figure 5.11: An exhaustive search of home placements for random graphs on eight data qubits shows that the Home Interaction By Sums heuristic provides a near optimal home placement (within 1% in all cases) for small graphs.

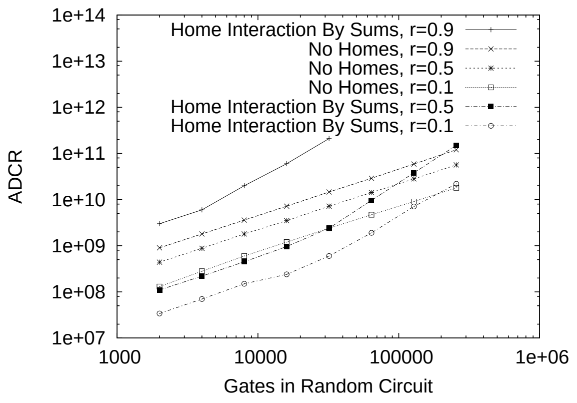


Figure 5.12: In addition to failing at large graph sizes, Home assignment fails for circuits with high Rent value because small clicks of highly interacting qubits simply don't exist. In this case, it is better not to enforce Homes.

Finally, Figure 5.12 shows the impact of the Rent parameter of the graph on our choice of heuristic. At very high Rent, qubit communication is so distributed and varied that small clicks of highly interacting qubits cannot be found. In this case, enforcing Homes is a bad idea, and it's better to allow qubits to roam across the datapath as they need to.

5.4 Home Assignment By Simulated Annealing

Deterministic mapping heuristics have the advantage of being relatively fast, while exhaustive search techniques become intractable very quickly as problem size grows. Between these two extremes lie probabilistic methods which perform some random and some deterministic traversal of the search space. Simulated annealing is a successful classical methodology which may be applied to our problem of Home assignment. We begin by explaining simulated annealing in the classical realm, then we apply it to our task.

5.4.1 Classical Simulated Annealing

Simulated annealing is an optimization heuristic based on an adaption of the Metropolis-Hastings algorithm, a Monte Carlo method to generate sample states of thermodynamic systems. It was first introduced by Kirkpatrick et al. in 1983 [38].

Simulated annealing is a global optimization approach that is used to find the optimum of some objective over large design spaces. The inspiration behind simulated annealing came from annealing in metallurgy, a technique involving heating and subsequent cooling of metals to allow metal structures to attain their lowest internal energy states. By analogy with this process, simulated annealing starts with an initial state in the design space, and initially allows for near-random transitions to "nearby" states. This simulates the behavior of metals when heated to high temperatures. As time proceeds, the transitions become more restricted to states that improve the optimization criteria. The restriction of the transitions is defined by a control parameter, called the temperature. As temperature decreases, it is more likely that only transitions that improve the objective be allowed.

5.4.2 Quantum Simulated Annealing

A valid state s_k for our problem consists of the assignment of a distinct Home gate location to each of the logical data qubits in our circuit. We now describe the key characteristics of simulated annealing as applied to our problem.

Objective Function: The Objective Function specifies the value of the optimization objective for a given complete assignment of Homes. We perform a full mapping and use the computed ADCR as our evaluation metric.

Temperature Function: The Temperature Function describes the temperature as a function of the iteration number. Common temperature functions include geometric and fractional schedules [46]. In this work, we use the function described by Koch [47] and tested extensively in [45]:

$$Temp(i) = \begin{cases} \frac{Temp(i-1)}{1+\delta\frac{Temp(i-1)}{\sigma_{i-L,i}}} & (i \ mod \ L) \ = \ 0 \\ Temp(i-1) & otherwise \end{cases} \tag{5.1}$$

where

$$\sigma_{i-L,i} = stddev\{ObjectiveFunc(s_k)|i-L \leq k \leq i\} \tag{5.2}$$

According to this function, the temperature only changes once every L steps. The extent of the temperature decrease is determined by two factors: δ, which is a constant multiplicative factor (set to 0.3 as in [45], and $\sigma_{i-L,i}$, which accounts for the standard deviation in the makespans produced in the last L steps. A high standard deviation means that the annealing has not stabilized yet, hence the temperature is only slightly decreased. When stabilization occurs, the temperature drops to near zero and the annealing procedure ends.

Probability Function: The Probability Function determines whether or not a cost-increasing transition is accepted. A normal probability function uses an increasing function of the $-\frac{\Delta c}{Temp}$ ratio. We use the following function from [45]:

$$Prob(\Delta c, Temp) = exp(-\frac{\Delta c}{Temp}) \tag{5.3}$$

This function leads to lower acceptance probabilities as either the cost difference Δc rises or as the $Temp$ gets lower.

Move Function: The Move Function defines adjacency in the search space and thus the valid transitions which may occur. We define a single move in the annealing process to be a swap of two qubits' Homes on the datapath. Since the number of qubits in the circuit need not be a power of two, nor even for that matter, the number of Home locations on our regular datapath may slightly exceed the qubit count. Once all qubits have been assigned a Home, the rest of the Home locations are filled in with dummy qubits. A swap between a data qubit and a dummy qubit is a valid move, but the rare instances of swaps between two dummy qubits are disregarded, as they do not change the value of the Objective Function.

Max and Min Temperature: An annealing procedure must start with a high initial probability of acceptance of transitions p_{max} and a low final acceptance rate p_{min}. We should choose the initial and final temperatures to achieve these probabilities. Using the Probability Function above, the temperature t for a fixed probability p is:

$$t = \frac{\Delta c}{ln(\frac{1}{p})} \tag{5.4}$$

Assuming that we expect a minimum cost change of Δc_{min} and a maximum of Δc_{max}, the initial temperature is set to:

$$t_{init} = \frac{\Delta c_{min}}{ln(\frac{1}{p})} \tag{5.5}$$

and the final temperature is set to:

$$t_{final} = \frac{\Delta c_{max}}{ln(\frac{1}{p})} \tag{5.6}$$

Both Δc_{min} and Δc_{max} are dependent on the size of the circuit in question.

5.4.3 Comparison with Previous Techniques

Figure 5.13 shows that Home Assignment by Simulated Annealing does as much as 1.4 times better on small random graphs than Home Interaction By Sums. However, since each step in the simulated annealing process requires that a full mapping be done in order to evaluate the objective function, it takes much longer to converge and doesn't achieve as much improvement.

Figure 5.14 shows the same data as Figure 5.13, but with a shorter x-axis for better resolution. For small enough problems, simulated annealing may be worth using in order to squeeze out a bit more performance out of the mapping.

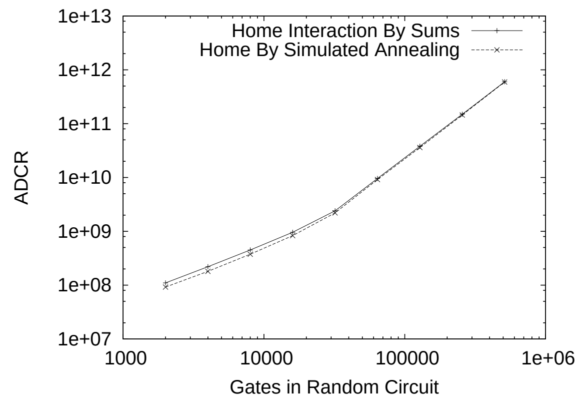


Figure 5.13: Simulated annealing wins by a factor of 1.4 on small graphs, but its longer running time causes it to do worse as graph size increases.

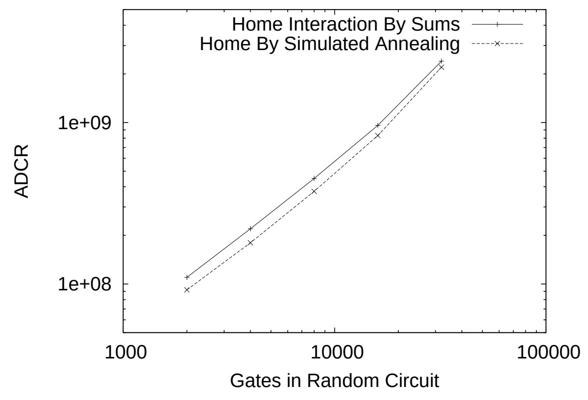


Figure 5.14: Same data as in Figure 5.13, but with a shorter x-axis for better resolution.

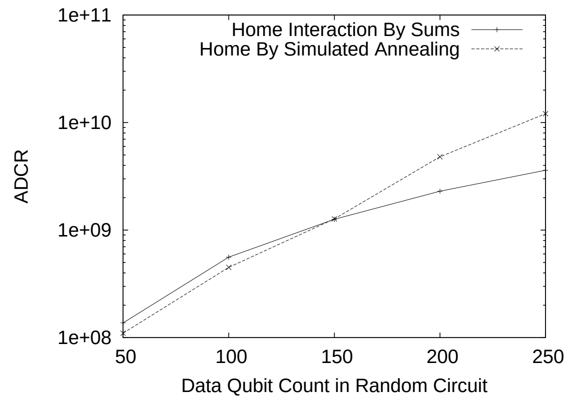


Figure 5.15: Simulated annealing starts failing on random graphs as qubit count goes past 150. Number of gates = 100 * number of qubits.

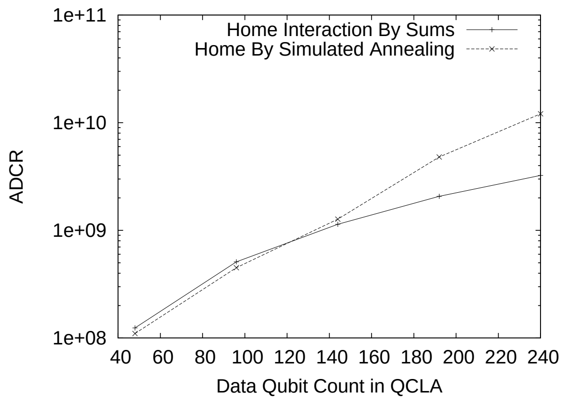


Figure 5.16: Simulated annealing works similarly on the QCLA as on random graphs, but the heuristic approach does relatively better on the highly structured adder, so the crossover point is lower. Number of gates = 100 * number of qubits.

While increasing the graph size increases the time to compute the objective function for simulated annealing, increasing the qubit count increases the size of the design space and thus has a much more drastic effect. As shown in Figure 5.15, the design space gets too large for feasible convergence past around 150 qubits. Thus, for problems with a few hundred qubits, the deterministic approach wins.

Figure 5.16 compares simulated annealing against the deterministic approach when applied to the Quantum Carry-Lookahead Adder, which is more structured in its communication pattern than random graphs. In this case, the crossover point is at fewer qubits since the data analysis approach has more regularity available to detect. The performance gain from simulated annealing on small problems is also slightly less than the 1.4 times gain on random graphs.

5.5 Reassigning Homes for Longer Runs

From Figure 5.9, we see that all Home-based algorithms start failing past a certain graph size. The reason is that communication patterns change, so a single Home assignment is inadequate for an entire program run. The solution is to reassign Homes periodically during the run.

Reassign Homes Every N Dataflow Graph Vertices: A simple solution is to pick a constant N and divide the overall graph into subgraphs of N vertices based on the centered graph priorities. Then we compute Home assignments individually for each subgraph. Figure 5.17 shows that the optimal value of N changes as qubit count changes. In each case, a period that is too large causes the algorithm to start performing poorly, as discussed earlier. However, a period that is too small results in

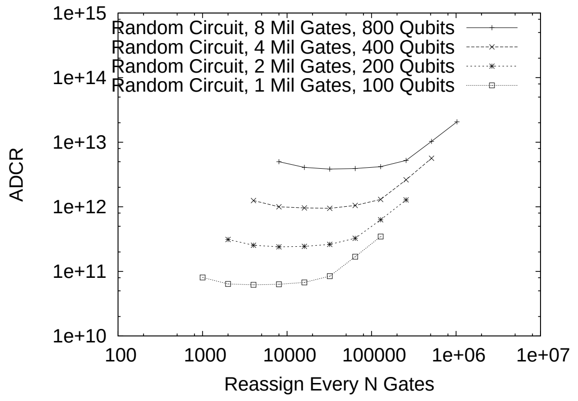


Figure 5.17: When reassigning Homes very N gates, the total qubit count affects the optimal period N. The gate count is varied simply to separate the lines, but it doesn't affect the optimal points.

unnecessary reshuffling of qubits and thus unnecessary use of the network, resulting in a worse ADCR.

Algorithm 1 Intelligent Home Reassignment Partitioning

```
1: set window size $W_s$
2: assign initial Homes based on the first $W_s$ two-qubit gates
3: initialize Interaction Counts
4: initialize Future counters to account for interactions in the first $W_s$ two-qubit gates
5: initialize graph traversal $T_f$ (future) and advance it $W_s$ two-qubit gates (without mapping)
6: begin mapping (any methodology)
7: for each mapped gate $G$ do
8:    if $G$ is a two-qubit gate then
9:       adjust interaction counts for $G$
10:      if $T_f$ not complete then
11:         poll $T_f$ for next gate $G_f$
12:         adjust Future interaction counts for $G_f$
13:      end if
14:   end if
15:   use counts to determine whether to reassign Homes
16: end for
```

Reassign Homes By Detecting a Change in Clicks: Alternatively, we could detect a change in the communication pattern and reassign Homes when it becomes substantially changed according to some metric. Algorithm 1 outlines the approach. As we perform the mapping, we keep a tally of interaction counts between each pair of qubits in a Future Window of size W_s gates. We set the window size equal to the optimal value from Figure 5.17 according to qubit count.

We perform our primary traversal of the dataflow graph as part of the mapping. We simultaneously perform an additional Future Traversal, T_f, which is W_s two-qubit gates ahead of the primary traversal. When a gate is mapped, its qubit interaction (if it is a two-qubit gate) is removed from the Future Window by decrementing the appropriate counter.

Using this information, we need to devise conditions under which to reassign Homes. For each qubit q, we compute each its *click* as follows:

1. Initialize the click C to the empty set

2. Sort the other qubits by interaction count with q within the Future Window: call this list L

3. If the sum of the interaction counts with qubits in C accounts for over 80% of q's total interaction count in the Future Window *or* if no qubits remain in L with interaction count with q that is greater than one, then we're done computing C

4. Otherwise, add to C the set of qubits in L with the highest interaction count, then remove them from L

5. Repeat Steps 3 and 4 until either condition in Step 3 is met

Interaction counts of one are ignored because they do not represent a significant enough impact on the communication pattern. Note that it is possible to devise a communication pattern such that $q0$ is in $q1$'s click, but $q1$ is not in $q0$'s, which is perfectly fine. Most of the time, however, the relationship is mutual.

We now define a couple of terms:

- Future Same (FS): The percentage of interactions in the Future Window in which the qubits are in each other's click. If the relationship is not mutual, it counts as half an interaction.

- Future Diff (FD): The percentage of interactions in the Future Window in which the qubits are *not* in each other's click.

Clicks are recomputed only when Homes are assigned. FS and FD are updated after each two-qubit gate is mapped, not just when Homes are reassigned. After each such update, Homes are reassigned if FD is greater than some percentage of FS, signifying that the communication pattern has changed sufficiently to warrant recomputation of Homes.

Rent Parameter	Optimal %
0.1	21
0.2	29
0.3	55
0.4	64
0.5	70
0.6	89
0.7	112
0.8	N/A
0.9	N/A

Table 5.2: Homes are reassigned when FD is great than some % of FS. The optimal value of % is highly dependent on the Rent parameter of the graph.

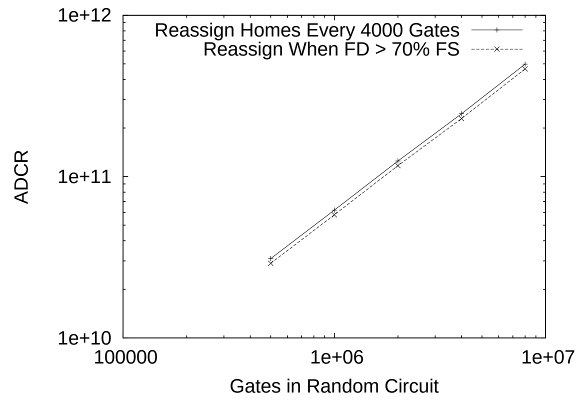


Figure 5.18: For Rent parameter 0.5, reassigning Homes when FD $>$ 70% of FS gives a 1.07 improvement over reassigning after every 4000 gates.

It turns out that the optimal value for this percentage parameter is independent of gate and qubit count but highly dependent on the Rent parameter of the dataflow graph. Table 5.2 shows optimal values for varying Rent values. At low Rent values, communication patterns tend to remain static, thus highly effective Homes may be assigned. At higher Rent values, it's not worth imposing the overhead of Home reassignment as often, since communication is more evenly distributed. At the highest Rent values, enforcement of Homes is detrimental, as shown in Figure 5.12.

We can now use these results to compare the FD/FS approach against periodic Home reassignment. From Table 5.2, we see that the optimal percentage for a Rent parameter of 0.5 is 70%. From Figure 5.17, we choose 4000 gates as the period for periodic reassignment. Figure 5.18, shows that we get an improvement in ADCR of 1.07 times under these conditions.

Figure 5.19 shows a similar comparison for Quantum Carry-Lookahead Adder

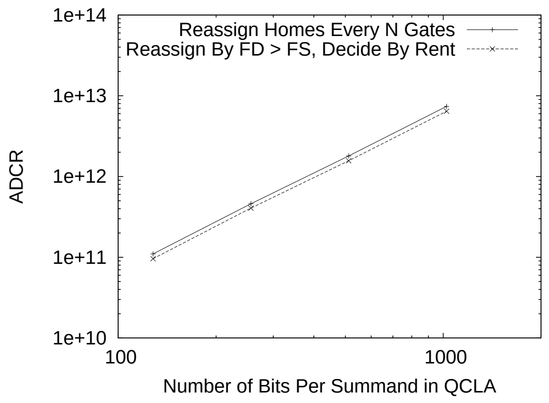


Figure 5.19: On the QCLA, Home reassignment using the FD/FS method with the optimal percentage value gives us more than 1.1 times improvement due to the circuit's less random communication patterns.

circuits of varying size. The static period is set based on qubit count, while the percentage value in the FD/FS method is set according to the Rent parameter. Due to the more structured communication patterns (than those found in random circuits), the adaptive method achieves up to a 1.15 times improvement over the static period method.

5.6 Summary of Mapping Techniques

The lessons learned in this chapter are threefold. First, assignment of Homes to qubits assists in localizing the movement of critical qubits, thus reducing the amount of expensive, teleportation-based communication on the critical path of execution. The end result is a significantly improved ADCR and thus a better final design for our datapath.

Second, reassignment of Homes is necessary as circuit size grows, since changing communication patterns necessitate reevaluation of qubit clicks. Reassignment done periodically works quite well, but in order to find the optimal reassignment period, we must consider the communication frequency and thus the Rent parameter of the target circuit. Third, assignment of Homes by simulated annealing provides a slight improvement over the deterministic approach at an extremely high cost in computation time. However, if this computation time is not a factor, simulated annealing does in fact provide the best result.

Chapter 6

Mapping Large Circuits

In previous chapters, we designed the basic components of our Qalypso architecture, including sample ancilla factories, the structure of compute regions and the basic subcomponents of the teleportation-based interconnect. We further developed tools to automatically map a circuit onto the malleable Qalypso datapath while optimizing our primary metric, ADCR. We are now ready to put all the heuristics together to map a very large, practical quantum circuit: factorization of a 1024-bit number.

Figure 6.1 shows a high level schematic of n-bit quantum factorization, which consists of two primary subcircuits: Modular Exponentiation and the Quantum Fourier Transform. The bulk of the circuit (over 99% of the gates) is in Modular Exponentiation, which has at its core an n-bit quantum adder. For this reason, we study n-bit quantum adder circuits first, which we will then use to fully implement factorization.

The purpose of this chapter is to demonstrate the overall contribution of this thesis as applied to a large practical circuit. We first describe the unoptimized mapping heuristic derived from prior work, followed by our fully optimized heuristic derived from the results in this work. We then apply the two heuristics to the adder circuits to find optimized adders. Finally, we apply the two heuristics to Shor's factoring algorithm to demonstrate our overall gain.

6.1 Unoptimized and Optimized Mappers

Our baseline mapper, hereafter referred to as *Unoptimized Mapper*, is the "Empty Loc Between the Qubits" algorithm introduced in Section 5.2.2. In summary, it is a direct application of classical priority list scheduling, with vertex priorities set according to the Centered Graph method and optimal gate location being any location with minimized Manhattan distance for communication. This baseline is a naive implementation of list scheduling assumed in prior quantum work.

Our fully optimized mapper, hereafter referred to as *Fully Optimized Mapper*, includes the optimizations illustrated in this work. Specifically, it involves the following:

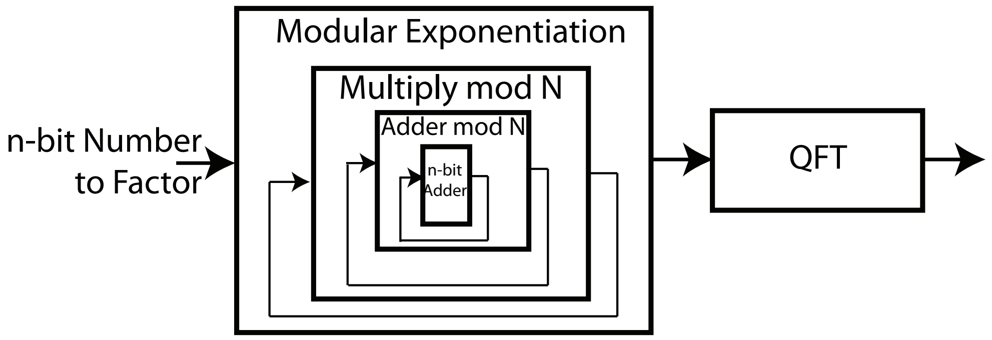


Figure 6.1: Shor's Factorization Algorithm: The majority of the work in Shor's factoring algorithm for an n-bit number is modular exponentiation, which has at its core repeated applications of quantum n-bit addition.

- Homes are assigned through the Interaction By Sums approach, which is the winning heuristic from Section 5.3.3 (since the circuits discussed in this chapter are far too large for the simulated annealing approach from Section 5.3.3).
- Homes are reassigned periodically according to the FD/FS heuristic described in Section 5.5.
- With Homes assigned, the Best Empty Loc When Penalizing SI evaluation metric from Section 5.2.2 is used to select a gate location for each logical gate.

The Fully Optimized Mapper cycles through the various Compute Region layout options discussed in Section 4.2 and varies support infrastructure as described in Section 4.3. For each such fixed datapath, it performs the mapping described above. The overall ADCR-optimal datapath and mapping is selected.

6.2 Quantum Addition Circuits

The quantum adder subcircuit comprises the bulk of Shor's factorization algorithm. For that reason, we will now evaluate our mapping heuristics on addition circuits before we tackle the larger factorization circuit.

6.2.1 Quantum Ripple Carry Adder

We evaluate the quantum ripple-carry adder (QRCA) [16] and the quantum carry look-ahead adder (QCLA) [17], constructing larger adders from smaller adder modules, similar to what is done with classical bit-serial adders, although we can have more than one instance of the smaller adder in the datapath. Figure 6.2 shows how

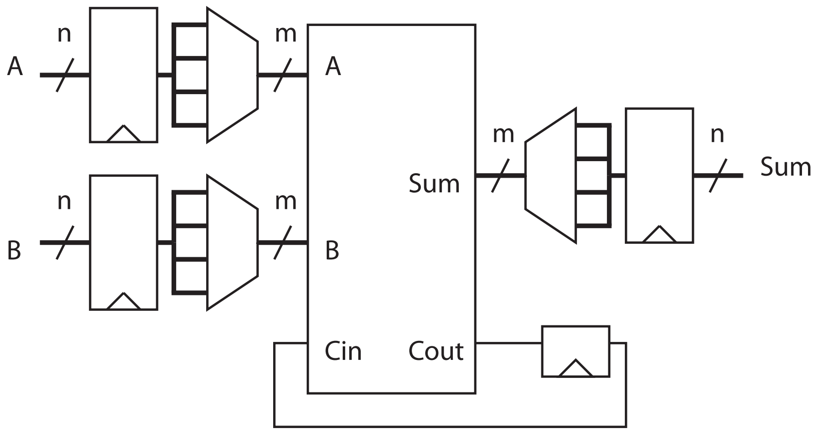


Figure 6.2: Our Quantum Ripple-Carry Adder (QRCA): An n-bit ripple-carry adder using $\frac{n}{m}$ cycles through an m-bit ripple carry adder. Carry out is registered and cycled back into subsequent iterations. Each ripple-carry block is similar to a classical ripple carry except that the carry bit must be inverted at the end to disentangle ancilla qubits.

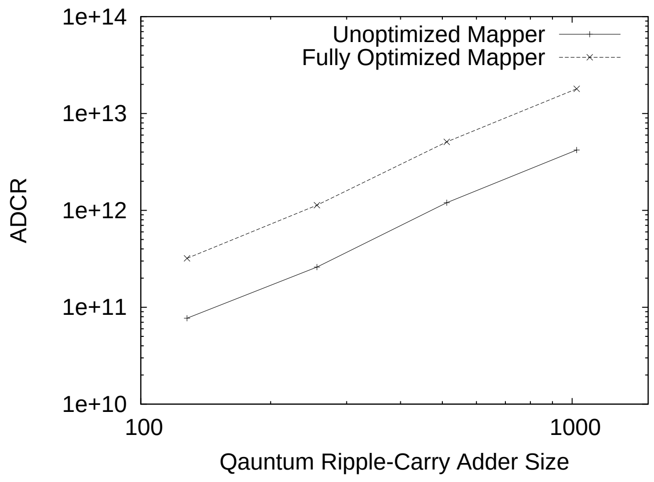


Figure 6.3: The Fully Optimized Mapper shows consistent improvement over the Unoptimized Mapper for Quantum Ripple-Carry Adder (QRCA) circuits of varying sizes.

an n-bit QRCA is constructed with multiple passes through a single m-bit sub-adder. Since we don't need dedicated memory regions in our datapath, the registers in Figure 6.2 represent idle qubits in gate locations, while the adder block represents active use of gate operations.

Figure 6.3 shows the impact of the mapping optimizations introduced in this work over the baseline direct implementation of classical priority list scheduling (the

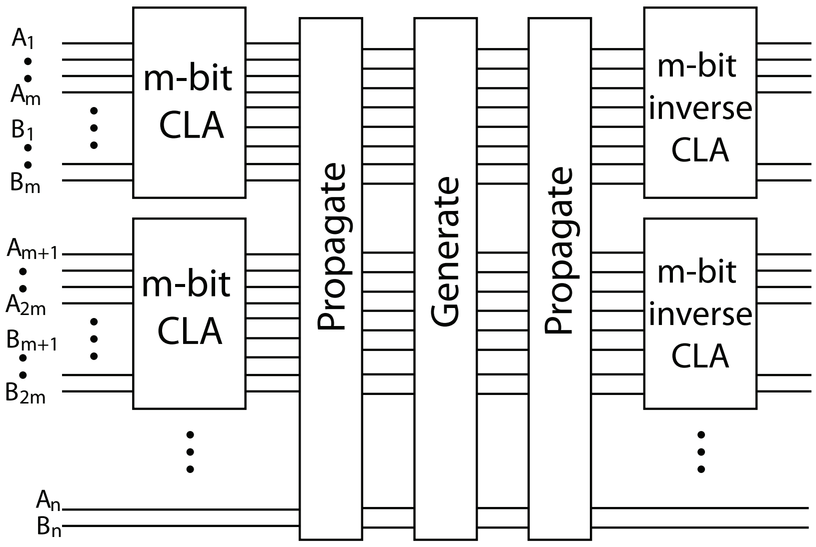


Figure 6.4: Our Quantum Carry-Lookahead Adder (QCLA): As with a classical CLA, the first few levels of the propagate and generate networks are built with uniform sized blocks. The logarithmic depth networks are completed in the blocks that span all bits. We must reverse the propagate and generate bits to disentangle them from the output.

Unoptimized Mapper) on QRCA circuits ranging from the 128-bit adder to the 1024-bit adder. The Fully Optimized Mapper shows a consistent improvement of more than a factor of four over the base case for these adders.

6.2.2 Quantum Carry Lookahead Adder

Similar to the QRCA in Figure 6.2, Figure 6.4 shows how an n-bit QCLA is constructed with smaller modules. The modular approach allows us to trade area for parallelism thus allowing us to construct optimal adder configurations. With both adders, the chief difference from the classical equivalent is that quantum circuits must necessarily be reversible, thus in each case there is a disentangling step at the end of each cycle.

The comparison between the Fully Optimized and Unoptimized Mappers for these circuits is shown in Figure 6.5. In this case, the Fully Optimized Mapper shows a gain of more than a factor of five across the range of circuit sizes, which is slightly greater than the gain achieved for the QRCA circuits, likely due to the greater parallelism inherent in the circuit and thus greater potential gain from good mapping decisions.

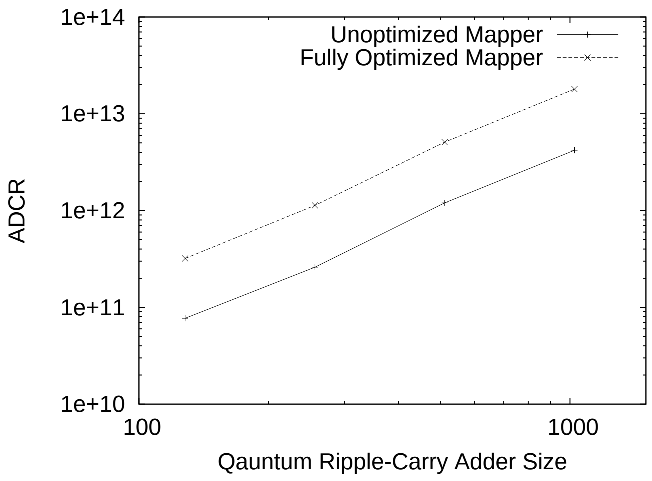


Figure 6.5: The Fully Optimized Mapper shows consistent improvement over the Unoptimized Mapper for Quantum Carry Lookahead Adder (QCLA) circuits of varying sizes.

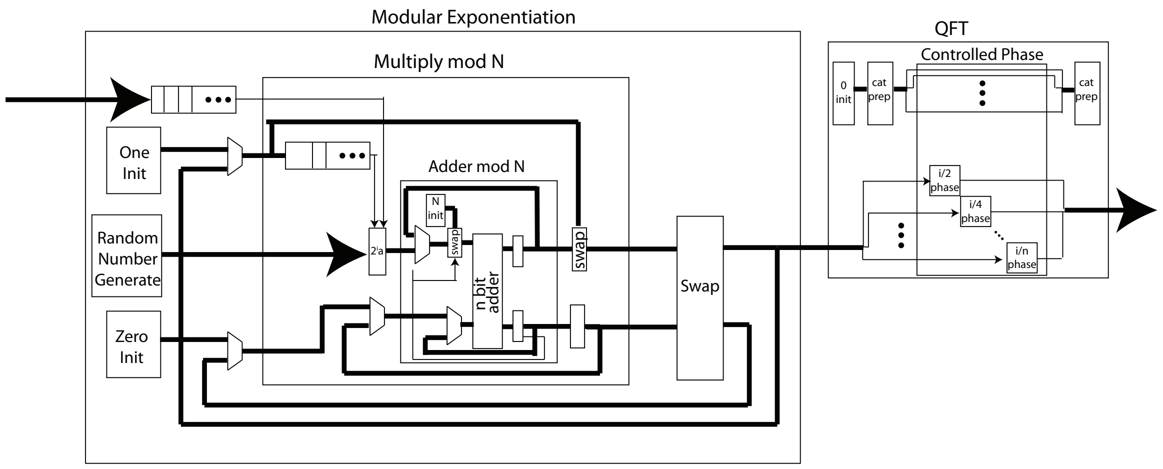


Figure 6.6: Shor's factorization consists of two major phases: modular exponentiation and Quantum Fourier Transform. The modular exponentiation circuit comprises the bulk of the execution time for Shor's factoring.

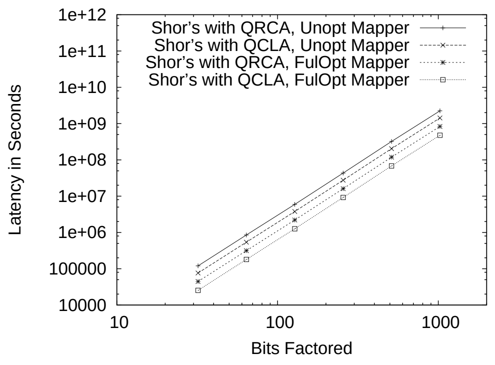


Figure 6.7: The Fully Optimized Mapper gains approximately a 3.0x speedup in latency on QCLA and 2.7x on QRCA for all sizes.

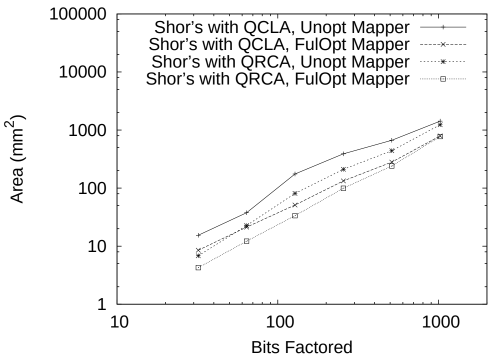


Figure 6.8: The Fully Optimized Mapper gains approximately 1.8x area reduction on QCLA and 1.6x on QRCA for all sizes.

6.3 Shor's Factorization Algorithm

We are now ready to compute the latency and area of optimal Shor's factoring circuits[1]. Figure 6.6 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 [63] and for the QFT, [26]. We implement and simulate two different versions of Shor's, one of which uses the 1024-bit QRCA (from Section 6.2.1) and the other of which uses the 1024-bit QCLA (from Section 6.2.2).

Figure 6.7 shows the overall gain in latency when using the mapper optimizations on both implementations of Shor's. In both cases, the speedup is approximately a factor of three. The greater parallelism and complexity of the Carry Lookahead Adder circuit (over the Ripple-Carry circuit) make room for slightly greater gain in latency.

Figure 6.8 shows a similar comparison with datapath area as the metric. The Fully Optimized Mapper provides an area gain of more than a factor of 1.5 in all cases, with slightly greater gain being achieved for the more complex QCLA implementation of Shor's, since there exists more opportunity for good (and poor) mapping choices.

6.4 Future Work

The simulation, optimization and layout techniques developed in this work get us closer to building realistic, tailored designs for quantum applications such as Shor's factorization algorithm. However, there are still many avenues for improvement that

[1]Our failure probability simulation is not yet up to handling circuits of the size of Shor's factoring. We leave that for future work.

require further investigation and development.

6.4.1 Laser Limitations

In ion trap technology, gate operations are performed by firing precisely timed and tuned laser pulses at the one or two qubits involved. Unfortunately, lasers are expensive and (relatively) large. As mentioned in Section 1.2.3, the proposed solution is to use an array of MEMS mirrors to redirect beams appropriately and to split beams to allow for SIMD application of gates. For the purposes of this work, we have assumed unlimited availability of lasers for gate operations.

A true implementation of this mechanism would require careful design of the mirror array and the associated control, as well as taking into account practical hindrances such as mirror realignment latency and accuracy. Further, there is ongoing study in the physics community concerning the number of different qubits which may be operated upon by a single laser beam. The intensity of the beam is the limiting factor, which could result in less available parallelism due to these laser restrictions.

6.4.2 Classical Control Hardware for Ballistic Movement

Another practical matter with an ion trap datapath is the application of pulse sequences to electrodes to achieve ballistic movement. While this process has been thoroughly demonstrated in laboratories, simultaneously controlling the movement of 100's or 1000's of physical qubits is another matter entirely.

Clearly, there is a great potential for SIMD application of such pulses since there are a limited number of different types of move operations (straight move along a channel, straight move through a 4-way intersection, etc.). Not only does the control circuitry need to be able to implement the ballistic movement specified by the mapping, but it needs to adapt to unexpected stalls, such as from a failed EPR pair purification, for instance.

6.4.3 Alternative Technologies

We have focused on ion trap quantum computers in this work because this technology has been studied and demonstrated adequately enough (on a small scale) to allow us to produce relatively realistic area and latency estimates for our designs. However, other technologies are still in contention, and it may be that different technological characteristics will be desirable for different applications.

Superconductor-based quantum operations are significantly faster than the same operations in ion traps, which provides good latency improvement potential but significantly exacerbates the control problem, since the control circuitry must operate at a much faster rate. Limitations imposed by this fact would need to be incorporated in the mapping and datapath design.

Qubits implemented in Liquid Helium-based technology suffer substantially more error during gate operations than during movement and idleness (even more so than in ion traps), suggesting that a teleport-based interconnect might be completely unnecessary in such a datapath. Other options worth exploring include solid state NMR and optical lattices.

6.4.4 Irregular Network Topologies

Throughout this work, we have assumed a uniform mesh grid teleport-based interconnect. This limitation of the design space has allowed us to more readily explore other variations, such as the mapping problem and Compute Region layout. However, we make no assertions that a mesh grid is the optimal network structure, and we leave this question open to further study.

6.5 Conclusion

In this work, we have presented a comprehensive toolset for laying out and scheduling a quantum datapath optimized for execution of a target quantum circuit. We have used this toolset to automatically construct and schedule such a datapath for Shor's factorization algorithm, with a resulting improvement in area-delay product of a factor of 5.4 over prior work.

We consider the quantum datapath to consist of three major components: the ancilla generation resources, the data qubit regions and the teleportation-based long distance interconnect. As an alternative to the simple ancilla generators assumed in previous approaches, we have presented designs for pipelined ancilla factories capable of producing a steady stream of encoded ancilla qubits at one or more designated output ports. These output ports allow us to reduce the movement of encoded ancillae as they travel to reach data qubits, thus eliminating both unnecessary error and congestion in ballistic channels.

We have presented the basic structure of data regions, which require gate locations and channels for ballistic movement, and we have also presented our network model which includes hardware designs of the basic components of the network. We have put these three datapath components together into our proposed architectural model, Qalypso, which includes some hand-crafted designs (such as for the ancilla factories), while not fully specifying each characteristic of the datapath. The chief benefit of Qalypso is that it reduces the layout design space to a manageable size while maintaining a sufficient degree of flexibility to allow us to tailor our quantum datapath to the circuit in question.

Finally, we have presented heuristics for finalizing a datapath from the Qalypso model to meet the needs of the target circuit while simultaneously optimizing the mapping of the circuit onto the datapath. We adapted priority list scheduling to this mapping problem and then improved upon this naive approach by introducing the notion of data qubit Homes, which discourage thrashing of critical qubits by localizing their movement as much as possible, resulting in a datapath and mapping appropriate for the target circuit.

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] D. Aharonov, V. Jones, and Z. Landau. A polynomial quantum algorithm for approximating the Jones polynomial. *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 427–436, 2006.

[3] S. Balensiefer, L. Kregor-Stickles, and M. Oskin. An evaluation framework and instruction set architecture for ion-trap based quantum micro-architectures. *Proc. 32nd Annual International Symposium on Computer Architecture*, 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] C.H. Bennett, D.P. DiVincenzo, J.A. Smolin, and W.K. Wootters. Mixed-state entanglement and quantum error correction. *Physical Review A*, 54(5):3824–3851, 1996.

[6] C.H. et al. Bennett. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Phys. Rev. Let.*, 70(13):1895–1899, 1993.

[7] DJ Bishop, CR Giles, and GP Austin. The Lucent LambdaRouter: MEMS technology of the future here today. *Communications Magazine, IEEE*, 40(3):75–79, 2002.

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

[9] D.C. Burger, T.M. Austin, and S. Bennett. *Evaluating Future Microprocessors: The SimpleScalar Tool Set.* University of Wisconsin-Madison, Computer Sciences Dept, 1996.

[10] AR Calderbank and P.W. Shor. Good quantum error-correcting codes exist. *Phys. Rev. A*, 54(2):1098, 1996.

[11] E. Chi, S.A. Lyon, and M. Martonosi. Tailoring quantum architectures to implementation style: a quantum computer for mobile and persistent qubits. *Intl. Symp. on Computer Architecture*, 2007.

[12] JI Cirac and P. Zoller. Quantum computing with cold trapped Ions. *Phys. Rev. Lett*, 74(20):4091–4094, 1995.

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

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

[15] S. Devadas, A. Ghosh, and K. Keutzer. *Logic synthesis*. McGraw-Hill, Inc. New York, NY, USA, 1994.

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

[17] 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.

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

[19] D.D. Thaker et al. Quantum Memory Hierarchies: Efficient Designs to Match Available Parallelism in Quantum Computing. *Intl. Symp. on Computer Architecture*, 2006.

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

[21] R. Ozeri et al. Hyperfine Coherence in the Presence of Spontaneous Photon Scattering. *Phys. Rev. Lett.*, 95:030403, 1995.

[22] S. Seidelin et al. Microfabricated surface-electrode ion trap for scalable quantum information processing. *Phys. Rev. Lett.*, 96(25):253003, 2006.

[23] T.S. Metodi et al. A Quantum Logic Array Microarchitecture: Scalable Quantum Data Movement and Computation. *Intl. Symp. on Microarchitecture*, 2005.

[24] W.K. Hensinger et al. T-junction ion trap array for two-dimensional ion shuttling, storage, and manipulation. *Appl. Phys. Lett.*, 88:034101, 2006.

[25] A.G. Fowler. Towards Large-Scale Quantum Computation. *Arxiv preprint quant-ph/0506126*, 2005.

[26] A.G. Fowler and L.C.L. Hollenberg. Scalability of Shor's algorithm with a limited set of rotation gates. *Phys. Rev. A*, 70(3):32329, 2004.

[27] Lenstra J.K. Graham R.L., Lawler E.L. and Kan A. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 1979.

[28] T. Grötker. *System Design with SystemC.* Kluwer Academic Publishers, 2002.

[29] L.K. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[30] S. Guide et. al. Implementation of the Deutsch-Jozsa algorithm on an ion-trap quantum computer. *Nature*, 421(6918):48–50, 2003.

[31] RN Hall, GE Fenner, JD Kingsley, TJ Soltys, and RO Carlson. Coherent Light Emission From GaAs Junctions. *Phys. Rev. Lett.*, 9(9):366–368, 1962.

[32] J. Hennessy, N. Jouppi, S. Przybylski, C. Rowen, T. Gross, F. Baskett, and J. Gill. MIPS: A microprocessor architecture. In *Proceedings of the 15th annual workshop on Microprogramming*, pages 17–22. IEEE Press Piscataway, NJ, USA, 1982.

[33] N. Isailovic, Y. Patel, M. Whitney, and J. Kubiatowicz. Interconnection Networks for Scalable Quantum Computers. *Intl. Symp. on Computer Architecture*, 2006.

[34] N. Isailovic, M. Whitney, Y. Patel, and J. Kubiatowicz. Running a quantum circuit at the speed of data. In *Intl. Symp. on Computer Architecture*, 2008.

[35] N. Isailovic, M. Whitney, Y. Patel, J. Kubiatowicz, D. Copsey, F.T. Chong, I.L. Chuang, and M. Oskin. Datapath and control for quantum wires. *ACM Transactions on Architecture and Code Optimization (TACO)*, 1(1):34–61, 2004.

[36] D. Kielpinski, C. Monroe, and D.J. Wineland. Architecture for a large-scale ion-trap quantum computer. *Nature*, 417(6890):709–711, 2002.

[37] J. Kim, S. Pau, Z. Ma, H. McLellan, J. Gages, A. Kornblit, and R. Slusher. System design for large-scale ion trap quantum information processor. *Quantum Information and Computation*, 5(7):515–537, 2005.

[38] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[39] L. Kreger-Stickles and M. Oskin. Microcoded Architectures for Ion-Tap Quantum Computers. In *Intl. Symp. on Computer Architecture*, 2008.

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

[41] R. Lipsett, C.F. Schaefer, and C. Ussery. *Vhdl: Hardware Description and Design.* Kluwer Academic Pub, 1989.

[42] C. Monroe, DM Meekhof, BE King, WM Itano, and DJ Wineland. Demonstration of a universal quantum logic gate. *Phys. Rev. Lett*, 75:4714–4717, 1995.

[43] HC Nägerl et. al. Laser addressing of individual ions in a linear ion trap. *Phys. Rev. A*, 60(1):145–148, 1999.

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

[45] Satish N.R. *Compile Time Task and Resource Allocation of Concurrent Applications to Multiprocessor Systems.* PhD thesis, University of California, Berkeley, January 2009.

[46] Salminen E. Orsila H. and Hamalainen T.D. *Simulated Annealing.* I-Tech Education and Publishing KG, 2008.

[47] Koch P. Strategies for realistic and efficeint static scheduling of data independent algorithms onto multiple digital signal processors. Technical report, The DSP Research Group, Institute for Electronic Systems, Aalborg University, Aalborg, Denmark, 1995.

[48] CE Pearson, DR Leibrandt, WS Bakr, WJ Mallard, KR Brown, and IL Chuang. Experimental investigation of planar ion traps. *Phys. Rev. A*, 73(3), 2006.

[49] J. Preskill. Fault-tolerant quantum computation. *Arxiv preprint quant-ph/9712048*, 1997.

[50] M. Riebe, H. Haffner, CF Roos, W. Hansel, J. Benhelm, GPT Lancaster, TW Korber, C. Becher, F. Schmidt-Kaler, DFV James, et al. Deterministic quantum teleportation with atoms. *Nature*, 429(6993):734–737, 2004.

[51] F. Schmidt-Kaler et. al. Realization of the Cirac–Zoller controlled-NOT quantum gate. *Nature*, 422(6930):408–411, 2003.

[52] N.A. Sherwani. *Algorithms for VLSI Physical Design Automation.* Kluwer Academic Publishers Norwell, MA, USA, 1995.

[53] 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.

[54] PW Shor. Algorithms for quantum computation: discrete logarithms and factoring. *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134, 1994.

[55] P.W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52(4):2493, 1995.

[56] PW Shor. Fault-tolerant quantum computation. *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, 1996.

[57] A. Steane. Multiple-Particle Interference And Quantum Error Correction. *Proceedings- Royal Society. Mathematical and physical sciences*, 452(1954):2551–2577, 1996.

[58] A.M. Steane. Space, Time, Parallelism and Noise Requirements for Reliable Quantum Computing. *Quantum Computing: Where Do We Want to Go Tomorrow?*, 1999.

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

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

[61] K.M. Svore, D.P. DiVincenzo, and B.M. Terhal. Noise Threshold for a Fault-Tolerant Two-Dimensional Lattice Architecture. *Arxiv preprint quant-ph/0604090*, 2006.

[62] D.E. Thomas and P.R. Moorby. *The Verilog Hardware Description Language.* Kluwer Academic Publishers, 2002.

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

[64] M. Whitney, N. Isailovic, Y. Patel, and J. Kubiatowicz. Automated Generation of Layout and Control for Quantum Circuits. In *Intl. Conf. on Computing Frontiers*, 2007.

[65] DJ Wineland, C. Monroe, WM Itano, D. Leibfried, BE King, and DM Meekhof. Experimental issues in coherent quantum-state manipulation of trapped atomic ions. *Arxiv preprint quant-ph/9710025*, 1997.

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

[67] C. Zalka. Simulating quantum systems on a quantum computer. *Mathematical, Physical and Engineering Sciences*, 454(1969):313–322, 1998.

[68] B. Zeng, A. Cross, and I.L. Chuang. Transversality versus Universality for Additive Quantum Codes. *Arxiv preprint arXiv: 0706.1382*, 2007.

[69] X. Zhou et al. Methodology for quantum logic gate construction. *Phys. Rev. A*, 62(5):52316, 2000.