A Performance Simulator for Quantitative Analysis of the Implications of Fault-Tolerance in Distributed Quantum Computer Architectures

Jeff Hussmann
Advisor: Jungsang Kim
Duke University

Abstract

Successfully performing a meaningful quantum computation will require protecting information from errors induced by faulty physical processes or spontaneous decoherence through the use of quantum error correcting codes and fault-tolerant circuit methodologies. I have built a performance simulation software package to analyze the overall performance implications of choices made in the design of these processes for a type of distributed quantum architecture. The software takes a set of inputs specifying the high-level organization of the computer, the code concatenation structure used to protect data, the circuit to be simulated, and the physical parameters of the computers basic operations. Its primary outputs are the execution time and probability of success of the overall computation. The flexible design of the software allows a wide variety of architectural issues in quantum computing to be easily explored.

I. Introduction

Quantum information processing is a model for computation in which information is stored in multi-state quantum mechanical systems and manipulated using carefully designed physical interactions to perform transformations of the state space [7]. The most exciting potential application of quantum computing is Shor's Algorithm, a procedure for factoring an integer in polynomial time, offering an exponential speed up over the fastest known classical algorithm [8]. Shor's Algorithm consists of selecting a random integer x relatively prime to the integer n to be factored, then using an inverse quantum Fourier transform to carry out a phase estimation procedure to determine the order of x with respect to n. From this order, a factorization of x can be determined with high probability [7]. Experimental implementations of Shor's Algorithm have demonstrated its practical feasibility on small scales by factoring the number 15 [12], but these experiments are not designed to be able to ramp up the size of the integer being factored to the point where the reduced computational complexity of the quantum algorithm translates into a performance improvement over current classical computers. In order to accomplish this, a fundamentally more scalable architecture is required. Additionally, quantum information will need to be encoded using quantum error correcting codes and operated on using fault-tolerant circuit constructions in order to protect information from error [10]. A complex and rapidly evolving array of schemes to accomplish this task exist, and choosing which of these schemes to use is an important and unique aspect of quantum computer architecture. Choices made in this area will interact with other architectural choices as well as technological limitations in an complicated way to determine the overall performance characteristics of a quantum computer. I have built a software simulator called Quipsim (quantum information processor simulator) to analyze this interaction for type of highly scalable distributed quantum computer architecture.

In part II, I will review basic principles of quantum error correction and fault tolerance. I will also present the ideas behind the distributed architecture being considered. In part III, I will present the fundamental design goals for the software. I will then provide an overview of its structure and an in-depth walk-through of its important processes. In part IV, I will present some simple results to demonstrate the capabilities of the software. In part V, I will explore potential future research applications of the software.

II. Background

1. Quantum Error Correction

Quantum error correction is needed to protect the integrity of quantum information from the potentially faulty physical processes used to prepare, store, and process the quantum information. In general, a quantum

error correcting code uses a number of physical qubits to encode a single logical qubit in such a way that some set of possible error operations can be reliably detected, distinguished, and corrected. Suppose that a code maps a set of k basis vectors for a 2^k-dimensional logical Hilbert space to a set $\{|\psi\rangle\}$ of k basis vectors for a 2^k-dimensional code subspace of the 2^n-dimensional physical Hilbert space. The quantum error correction condition gives a necessary and sufficient condition for the code space $\{|\psi\rangle\}$ to be able to correct a set of errors $\{E\}$. Specifically, if

$$\langle\psi_i|E_a^\dagger E_b|\psi_j\rangle = C_{ab}\delta_{ij}$$

for any $E_a, E_b \in \{E\}$ and any $|\psi_i\rangle, |\psi_j\rangle \in \{|\psi\rangle\}$, then the code defined by $\{|\psi\rangle\}$ is capable of correcting any arbitrary linear combination of errors in $\{E\}$ [6, 2]. Intuitively, this reflects the facts that the results of different errors acting on codewords must be orthogonal in order to reliably distinguish between different errors and that no information must be gained about the actual state of the codeword in order to preserve superposition states.

A broad class of codes can be defined by the stabilizer formalism developed by Gottesman in which a code space is described implicitly by giving a group of operators that 'stabilize' the code space [2]. Let $\mathcal{G}_1$ (the Pauli group on 1 qubit) be the group of single qubit operators generated by I, the Pauli matrices X, Y, and Z, and the multiplicative factors of $\pm i$, and let $\mathcal{G}_n$ be the tensor product of n copies of $\mathcal{G}_1$. The weight of an element in $\mathcal{G}_n$ is the number of non-I terms it contains. Given a code space $\{|\psi\rangle\}$ that encodes k logical qubits in n physical qubits, let S be the set of all operators in $\mathcal{G}_n$ that have $|\psi_i\rangle$ as an eigenvector with an eigenvalue of $+1$ for all $|\psi_i\rangle \in \{|\psi\rangle\}$. If $\{|\psi\rangle\}$ is nontrivial, then S is an abelian subgroup of $\mathcal{G}_n$ that can be completely described by $n-k$ generators. It can be shown that if E_aE_b anticommutes with some element of S for all $E_a, E_b \in \{E\}$, then the set of errors $\{E\}$ satisfy the quantum error correction condition. An error in S is also trivially correctible. Since all operators on n qubits can be expressed as linear combinations of elements of $\mathcal{G}_n$, this means that if the minimum weight of an element that anticommutes with a generator of S is d then the code can correct arbitrary errors on up to $\lfloor d/2 \rfloor - 1$ qubits.

Calderbank-Shor-Steane (CSS) codes are an important subclass of stabilizer codes with attractive fault-tolerance properties [7, 2]. CSS codes are based on classical linear codes. As the name suggests, the set of codewords of a linear code is closed under binary addition. Given a linear code C, the dual of C is the set of all codewords that are orthogonal to every codeword in C and is denoted $C^\perp$. Given two classical linear codes C_1 and C_2 each capable of correcting t errors such that $C_2^\perp \subset C_1$, the codewords of the quantum CSS code of C_1 over $C_2^\perp$ are defined by

$$\left\{ \sum_{y \in C_2^\perp} |x+y\rangle \mid x \in C_1 \right\},$$

where global multiplicative factors have been ignored. There are as many distinct codewords in the code as there are cosets of C_1/C_2, and each codeword consists of a linear superposition of all members of a given coset. The error correcting properties of this construction follow elegantly from the restrictions placed on C_1 and C_2 and will be verified below in the discussion of fault-tolerant error correcting procedures for CSS codes.

2. Fault Tolerance

In order to achieve useful quantum computation, we need to be able to protect information from error throughout the lifetime of its role in the computation. To do this, fault-tolerant procedures are needed to limit the scope of the contamination introduced by any single error event. Ideally, this will prevent the total number of physical qubits in error in each logical block from cascading beyond the ability of the code to correct the errors. Rigorous definitions can be given for the conditions processes must satisfy to be considered fault tolerant [4]. Focusing on codes that only correct errors on a single qubit simplifies the scope of this discussion considerably. We will call a process fault tolerant if it can begin with an error-free logical block, suffer a single isolated error anywhere in the course of the process, and produce a logical block that contains only one qubit in error [3]. (Of course, we also assume that the process achieves its desired goal.) A logical operation is said to have a transversal implementation for a code if it can be carried out by physical operations that act on only one physical qubit per block, as in the transversal CNOT implementation in Fig. 1. Transversal operations are ideal from a fault-tolerance standpoint since they can trivially be seen to satisfy the definition of fault-tolerance while requiring a minimally complicated construction. Operations that do not admit to transversal implementations in a code may require substantial work to achieve fault-tolerance. The stabilizer formalism allow us to determine which operations can be implemented transversally in a given code [3]. If a state $|\psi_i\rangle$ is stabilized by the operator S_j, then for an arbitrary operator U, the state $U|\psi_i\rangle$ is stabilized by $USU^\dagger$. Using this property allows us to find

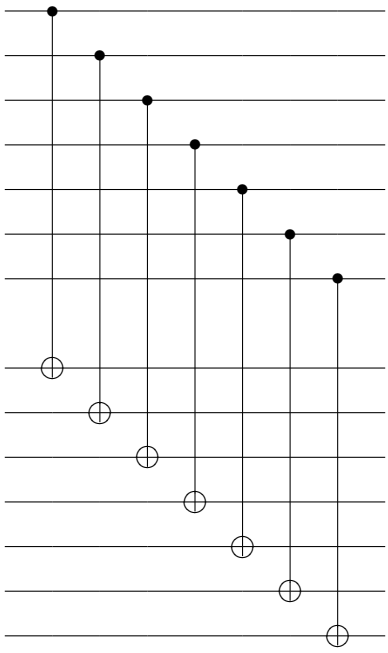

Figure 1: Transversal CNOT implementation

transversal implementations for gates by determining how the desired gate transforms the stabilizer of the code. In situations where transversal implementations are not possible, stabilizer manipulations provide insight into possible non-transversal designs. This will be seen in the discussion of fault-tolerant Toffoli gates for CSS codes discussed below.

We want to be able to perform universal computation in a fault tolerant manner. Clearly, this will require fault tolerant implementations of a universal set of gates. Unfortunately, it appears that no stabilizer code has a universal set of transversal gates [13]. This necessitates the development of non-transversal fault-tolerant constructions for some set of gates which completes universality. There are many possible universal sets of gates to work with [7], and schemes for completing fault-tolerant implementations for several of these sets are outlined in [11]. For experimental and theoretical convenience, we will choose the universal set obtained by taking the Clifford group (consisting of the Pauli group, CNOT, H, and S gates [14]) together with the Toffoli gate.

3. State Preparation and Error Correction Procedures

In order to protect information using quantum error correcting codes, the constituent qubits which make up a logical qubit must be prepared into a codeword state. Once a prepared logical qubit is undergoing a computation, error correction procedures must be periodically carried out on it. There is considerable overlap between these two processes.

In the case of a general stabilizer code, the fact that measuring an operator on a state projects the state onto an eigenvector of the operator, a logical $|0\rangle$ or $|1\rangle$ state can be prepared by measuring the generators of a stabilizer code and a logical X or Z on a set of n physical qubits, followed by controlled application of anticommuting correction operations when a -1 eigenstate is measured‘[7]. An error correction procedure can be carried out by simply omitting the logical X or Z measurement. An example of state preparation via stabilizer measurement is shown in Fig. 2.

Measuring an operator on a state consists of collapsing the state into an eigenvector of the operator and receiving an indication of which eigenvector has been collapsed to. We are particularly interested in measuring unitary operators whose eigenvectors have eigenvalues of ± 1 since the Pauli matrices fall into this category and therefore any stabilizer generator which consists of a tensor product of Pauli matrices does as well. To measure a unitary operator U having eigenvectors of $|u^+\rangle$ with an eigenvalue of 1 and $|u^-\rangle$ with an eigenvalue of -1 on

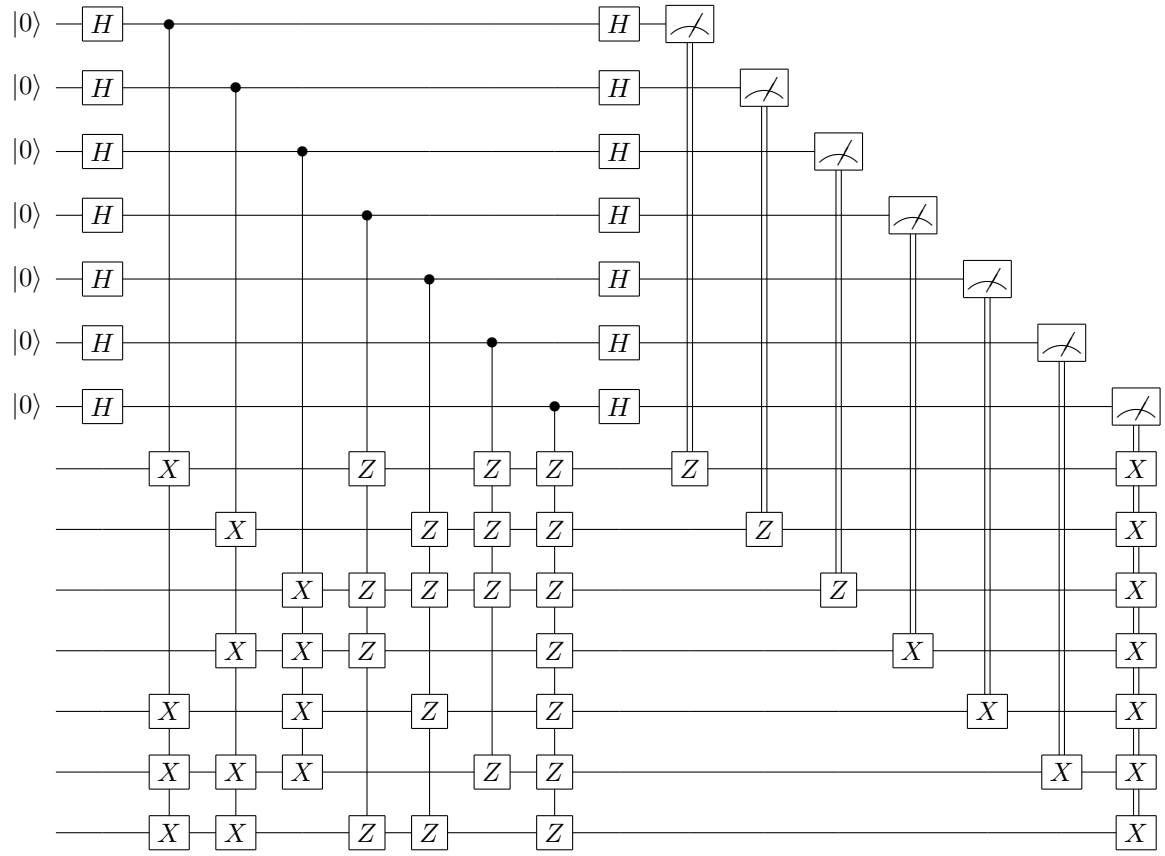

Figure 2: Steane code state preparation by stabilizer measurement

a state $|\psi\rangle$, we can use the circuit given in Fig. 3.

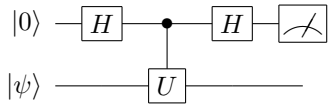

Figure 3: Measuring an operator

To see how this works, note that before any operations are done, the two qubit system is in the state $|0\rangle|\psi\rangle$. Since U is unitary and $|u^+\rangle$ and $|u^-\rangle$ have distinct eigenvalues, the two eigenstates are orthogonal and therefore form a basis for the two-dimensional state space of $|\psi\rangle$. This means that

$$|\psi\rangle = \alpha\,|u^+\rangle + \beta\,|u^-\rangle \tag{1}$$

for some α and β. Since the first Hadamard gate takes the ancilla from $|0\rangle$ to $|0\rangle + |1\rangle$, the overall state of the two qubits following the controlled-U operator is given by

$$|0\rangle\left(\alpha\,|u^+\rangle + \beta\,|u^-\rangle\right) + |1\rangle\,U\left(\alpha\,|u^+\rangle + \beta\,|u^-\rangle\right). \tag{2}$$

By the definitions of $|u^+\rangle$ and $|u^-\rangle$, this simplifies to

$$|0\rangle\left(\alpha\,|u^+\rangle + \beta\,|u^-\rangle\right) + |1\rangle\left(\alpha\,|u^+\rangle - \beta\,|u^-\rangle\right). \tag{3}$$

Grouping by common second qubit state gives

$$\alpha\left(|0\rangle+|1\rangle\right)\left|u^{+}\right\rangle+\beta\left(|0\rangle-|1\rangle\right)\left|u^{-}\right\rangle, \tag{4}$$

and applying the final Hadamard gate to the first qubit transforms this to

$$\alpha\,|0\rangle\left|u^{+}\right\rangle+\beta\,|1\rangle\left|u^{-}\right\rangle. \tag{5}$$

The net effect of the circuit prior to measuring the ancilla is therefore to entangle the component of ψ that is stabilized by U with the $|0\rangle$ component of the ancilla and the component of ψ that is multiplied by -1 when operated on by U with the $|1\rangle$ component of the ancilla. Measuring the ancilla in the computational basis therefore collapses $|\psi\rangle$ into an eigenstate of U and indicates which state this is.

If U is a multi-qubit operator (say, for instance, that $|\psi\rangle$ is a logical qubit and U is a stabilizer generator of the code $|\psi\rangle$ is encoded in), this process is clearly not fault tolerant since a single error on the ancilla could propagated through the controlled-U operation to affect several qubits in $|\psi\rangle$ [7]. Somewhat more subtly, a single error anywhere in this circuit could result in an incorrect value being reported by the actual measurement. This incorrect value could then be used to incorrectly execute classically controlled operations on an potentially large number of qubits, effectively propagating the single initial error to multiple locations'[4]. Fixing these problems incurs considerable expense. To avoid the first problem, the single ancilla qubit and its Hadamard gate can be replaced by n qubits prepared in a maximally entangled 'cat' state $|00...0\rangle+|11...1\rangle$. Since this preparation is itself inherently non-fault-tolerant, it is followed by a verification process in which the parity of a pair of qubits are compared for some subset of all pairs. To avoid the second problem, the entire process is repeated three times and a majority vote of the measurement results is taken to be the actual measured value. In order for this value to be incorrect, more than one error will have needed to occur so fault-tolerance is preserved.

The structure of CSS codes (specifically, the fact that codewords are superpositions of states which are themselves codewords of classical error correcting codes) allows for a more efficient fault-tolerant error correction scheme to be used for them [9]. Suppose that we wish to correct errors in an encoded state $|\psi\rangle=\alpha|\overline{0}\rangle+\beta|\overline{1}\rangle$. First, prepare an ancilla state in the logical superposition $|\overline{0}\rangle+|\overline{1}\rangle$. Then use $|\phi\rangle$ as the control for a CNOT operation with the ancilla as the target as shown in Fig. 4.

Viewing the initial ancilla state as $\left|\overline{0}\right\rangle+\left|\overline{1}\right\rangle$ allows us to see that the logical CNOT gate has no effect on the

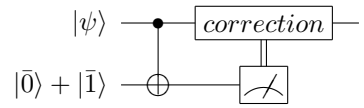

Figure 4: Correcting bit flip errors in CSS codes

state. Recalling the definition of CSS codes, however, this state can also be viewed as a superpostion of every codeword of C_1. Transversally measuring each qubit in the ancilla will therefore give an arbitrary codeword from C_1 without providing any information about $|\psi\rangle$ and therefore without collapsing any logical superposition $|\psi\rangle$ is in. The error-correcting properties of this process arise from the fact that any bit flip errors in $|\psi\rangle$ will propogate through the CNOT gate and appear as bit flip errors in the measured C_1 codeword. Since C_1 is a classical code capable of correcting up to t errors by construction, we can perform classical error correction on C_1 and learn which bits of $|\psi\rangle$ have been flipped. Applying an X operator to each of the indicated bits will therefore correct up to t bit flip errors. Following this, a similar construction, shown in Fig. 5 can then be applied to correct phase errors.

To understand how this works, note that since applying a transversal Hadamard gate to an n-qubit state $|y\rangle$ gives

$$H|y\rangle=\sum_{z\in\{0,1\}^n}(-1)^{y\cdot z}|z\rangle,$$

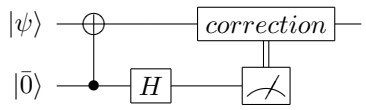

Figure 5: Correcting phase flip errors in CSS codes

applying a transversal Hadamard gate to the logical zero state of a CSS code gives

$$\begin{aligned} H|\bar{0}\rangle &= H \sum_{y \in C_2^\perp} |y\rangle \\ &= \sum_{z \in \{0,1\}^n} \sum_{y \in C_2^\perp} (-1)^{y \cdot z} |z\rangle. \end{aligned}$$

To simplify this sum, note that if $z \in C_2$, then $y \cdot z = 0$ for all $y \in C_2^\perp$ by definition so that

$$\sum_{y \in C_2^\perp} (-1)^{y \cdot z} = |C_2^\perp|.$$

If $z \notin C_2$, on the other hand, then there is at least one $y^* \in C_2^\perp$ such that $y^* \cdot z = 1$. For every $y \in C_2^\perp$ such that $y \cdot z = 0$, we can produce a unique $y + y^* \in C_2^\perp$ such that

$$(y + y^*) \cdot z = y \cdot z + y^* \cdot z = 1.$$

This process partitions $C_2^\perp$ evenly into a set whose elements are orthogonal to z and a set whose elements are not, so exactly half of the elements of $C_2^\perp$ are orthogonal to z and

$$\sum_{y \in C_2^\perp} (-1)^{y \cdot z} = 0.$$

These two cases together imply that the overall sum simplifies to

$$H|\bar{0}\rangle = \sum_{z \in C_2} |z\rangle.$$

If $|\psi\rangle$ is an error-free state, using this ancilla as the control for a CNOT with $|\psi\rangle$ as the target will not affect either state. If $|\psi\rangle$ contains any phase errors, however, they will propogate backwards through the CNOT where they will be transformed by the transversal Hadamard gate into bit flip errors. Measuring the final ancilla state transversally will therefore give an arbitrary codeword of C_2 with bit flips wherever phase flips occured in $|\phi\rangle$. Classical error correction can then be applied to the measured codeword and Z gates applied to the indicated error locations to correct up to t phase errors. As discussed above, this ability to simulaneously correct bit- and phase-flip errors on up to t qubits implies the ability to correct arbitrary errors on up to t qubits.

4. Fault Tolerant Toffoli Gate Construction

CSS codes offer transversal implementations of the entire Clifford group but unfortunately not of Toffoli gates [3]. This necessitates an alternative construction to fault-tolerantly perform Toffoli gates. A systematic approach to developing such constructions consists of first creating well-known circuits to teleport data qubits onto blank ancilla and then perform the desired gate on the ancilla. The desired gate is then swapped back through the teleportation circuit according to commutation relations until the problem has been reduced to fault-tolerant preparation of a specific, hopefully more tractable ancilla state [14]. In the case of a Toffoli gate, this produces the circuit shown in Fig. 6.

In order for this construction to be completely fault-tolerant, we need to be able to fault-tolerantly the initial ancilla state isolated in Fig. 7. Since we know how to fault-tolerantly measure stabilizer operators, we want to

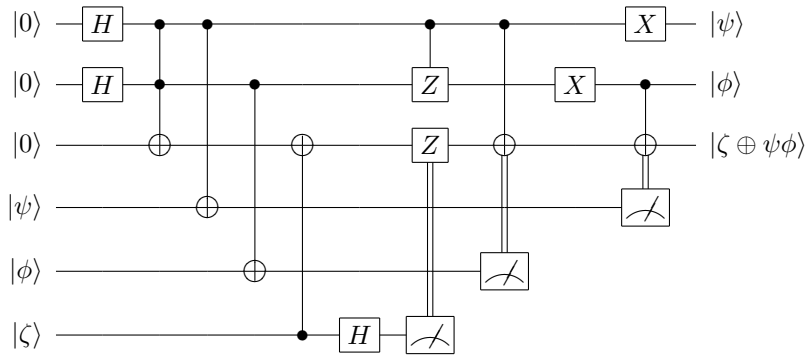

Figure 6: Fault-tolerant Toffoli construction

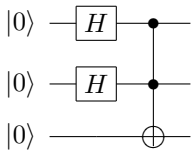

Figure 7: Ancilla state for fault-tolerant Toffoli process

decompose this circuit into an equivalent process consisting of measurements of stabilizers. Recall that if an operator S stabilizes a state $|\psi\rangle$, then applying a unitary operator U to $|\psi\rangle$ gives

$$U|\psi\rangle = US|\psi\rangle = USU^\dagger U|\psi\rangle,$$

so the state $U|\psi\rangle$ is stabilized by $USU^\dagger$. We start with the state $|000\rangle$ which is clearly stabilized by Z_1, Z_2, and Z_3, where subscripts indicate which qubits a single qubit gate acts on. Let U be a Toffoli gate. Then the state we want to prepare is $UH_1H_2|000\rangle$. To find stabilizers of this state, we conjugate the initial stabilizers by these operations and use the commutation relations $H_iZ_i = X_iH_i$, $H_iZ_j = Z_jH_i$, $UX_1 = X_1CNOT_{2,3}U$, $UX_2 = X_2CNOT_{1,3}U$, and $UZ_3 = Z_3CZ_{1,2}$ to give

$$\begin{aligned}
S_1 &= UH_1H_2Z_1H_2^\dagger H_1^\dagger U^\dagger \\
&= UX_1H_1^2H_2^2U^\dagger \\
&= X_1CNOT_{2,3}UU^\dagger \\
&= X_1CNOT_{2,3} \\
S_2 &= UH_1H_2Z_2H_2^\dagger H_1^\dagger U^\dagger \\
&= X_2CNOT_{1,3} \\
S_3 &= UH_1H_2Z_3H_2^\dagger H_1^\dagger U^\dagger \\
&= Z_3CZ_{1,2}
\end{aligned}$$

Measuring these operators would produce the desired ancilla state. To minimize the number of two-qubit operators which must be measured, however, note that

$$UH_1H_2|000\rangle = |000\rangle + |010\rangle + |100\rangle + |111\rangle$$

and

$$Z_1UH_1H_2|000\rangle = |000\rangle + |010\rangle - |100\rangle - |111\rangle.$$

Since Z_1 anticommutes with S_1, measuring S_1 on the state $UH_1H_2\left|000\right\rangle + Z_1UH_1H_2\left|000\right\rangle$ will produce the desired ancilla state. This intermediate state sums to

$$\left|000\right\rangle + \left|010\right\rangle = \left|0\right\rangle\left(\left|0\right\rangle + \left|1\right\rangle\right)\left|0\right\rangle$$

and is clearly stabilized by Z_1, X_2, and Z_3. We can therefore simply measure these three single qubit operators followed by S_1. The entire preparation (short of repetition and consensus-taking) is shown in Fig. 8.

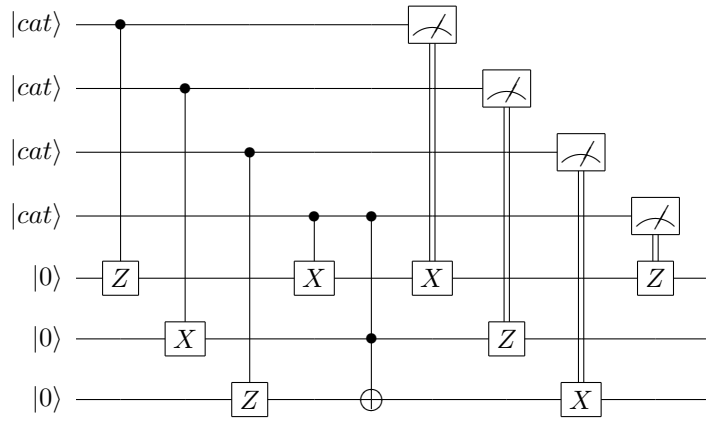

Figure 8: Fault-tolerant preparation of Toffoli ancilla state

5. Distributed Quantum Computer Architecture

The fundamental architecture being considered by Dr. Kim's group and therefore being targeted for simulation attempts to offer a high degree of scalability by separating a quantum computer into a set of distinct elementary logic units (ELUs) each containing a physically realizable number of qubits. A reasonable lower bound on the number of physical qubits each ELU must contain will be dictated by the coding scheme used, since the ability to store at least one and almost certainly several logical qubits in a single ELU will be highly desirable. Each ELU will possess the physical hardward necessary to carry out a universal set of quantum gates between any local set of its physical qubits. The computer as a whole will consist of a network of ELU's optically linked by a crossconnect switch capable of creating an optical connection between any arbitrary pair of ELU's. Using this optical connection, entangled states can be created which are distributed between different ELUs. A diagram of this architecture is shown in Fig. 9.

A classical microcontroller/microprocessor will interact with the network of ELUs to control the physical operations used to process qubits and to carry classical information between ELUs. The only quantum communication allowed between ELU's, however, is the aforementioned optical distribution of entangled pairs of qubits. Using the principle of quantum teleportation, a CNOT gate between remote ELUs can be performed using only these resources of shared entangled pairs and classical communication [14]. The construction for accomplishing this is derived in much the same way as the fault-tolerant Toffoli construction above. A teleportation circuit is first created from the two data qubits involved in the CNOT gate to be performed onto ancilla qubits. A CNOT is then added between the final ancilla states and commuted backwards through the circuit. In the resulting circuit shown in Fig. 10, the required ancilla state is precisely the shared entangled state that the optical network connecting ELUs is capable of generating. All processing beyond the creation of this state consists of local operations between qubits on the same ELU or classical communication between ELUs.

Armed with this remote CNOT gate, examination of the fault-tolerant Toffoli gate construction in Fig. 6 above reveals that a Toffoli gate can be performed between qubits distributed across three different ELUs simply by replacing every CNOT in the circuit with the remote form. Note, however, that the presence of physical Toffoli gates in the fault-tolerant ancilla preparation required by this circuit forces this preparation to take place inside a single ELU.

III. Software Design

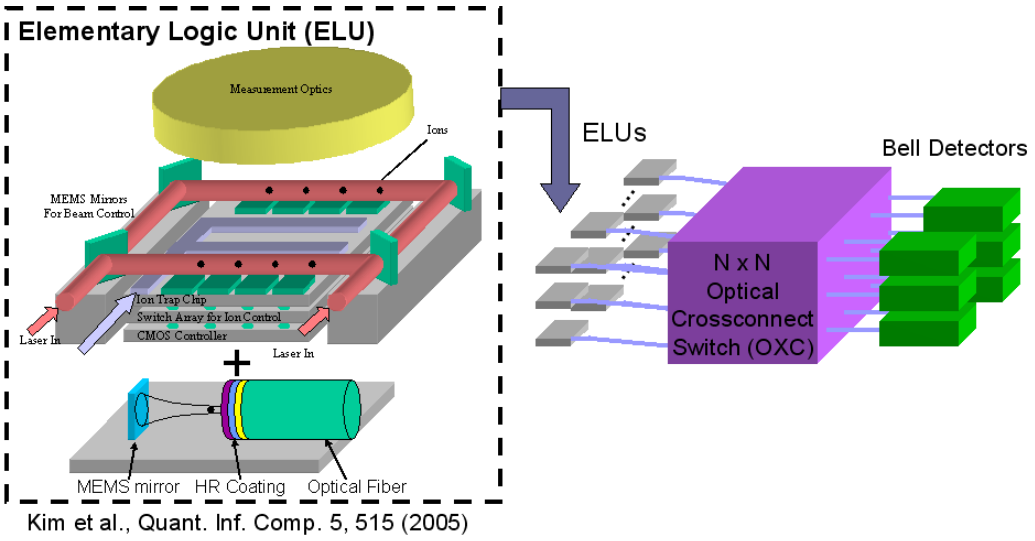

Figure 9: Distributed Quantum Computer Architecture - ELUs and entanglement generation optics

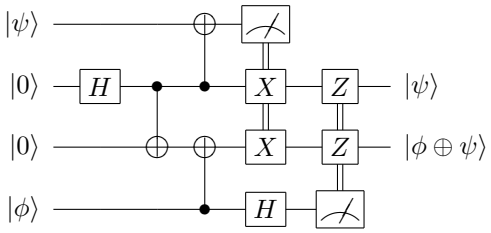

Figure 10: Remote CNOT construction

1. Design Goals

The software's purpose is to take a user-specified high-level quantum circuit description, code concatenation structure, and error correction strategy, appropriately decompose the high-level operations into their component physical operations, map these physical operations onto a representation of a distributed quantum computer built according to user-controlled parameters, then simulate the execution of these operations to generate execution time and probability-of-success performance characteristics of the overall design.

The two motivating design goals are flexibility and performance. It is desirable for a wide range of error correcting code design choices and architectural design choices to be easily varied by the end user. It is clearly also desirable for the simulator to operate efficiently in order to allow simulation of large and complicate circuits on a reasonable time scale. The software is implemented in C++ in order to achieve a balance between these goals. The software consists of a central framework representing the core distributed architectural paradigm along with an extensible set of modules defining the processes associated with a given quantum error correcting code. This modular design allows different options for processes associated with error correcting code design to be easily added and modified. It also enables recursive resolution of high level gates into component physical operations in a way that can be completely transparent to the end user.

2. High-level Organization

The software consists of four major components.

- **Quipsim master controller:** This is responsible for directing execution of a simulation. To begin any

simulation, it creates an instance of the Computer class described below according to user-specified architectural parameters and code concatenation choices. It then creates an array of Qubit objects corresponding to highest level logical qubits in the theoretical description of the circuit being simulated. It directs the Computer object to map these logical qubits onto actual physical qubits. It then breaks the circuit to be simulated into appropriate function calls on this array of Qubit objects. It also fills in implied error correction steps in the subset of all gaps in the circuit description chosen by the user. At the end of the simulation, it gathers performance data and presents formatted output to the user.

- **Computer class:** Objects of this class represent an entire distributed computer. A Computer object maintains an array of pointers to the ELU objects that make up the Computer. To maintain flexibility, a pointer-to-pointer-to-ELU is used to store this array rather than a fixed array of pointers to ELUs [5]. Upon Computer creation, the user-specified number of ELU objects are instantiated and linked to the Computer. The Computer is responsible for directing the placement of the sets of physical qubits that make up logical qubits onto its component ELUs. It is also responsible for scheduling computer-wide access to entanglement generation operations as discussed in section 4 below.

- **ELU class:** Objects of this class represent an elementary logic unit node in the a distributed computer. Each ELU represents its component physical qubits internally as an array of pointers to Qubits. This is declared as a pointer-to-pointer-to-Qubit and sized appropriatly upon instantiation of the ELU. ELU objects are responsible for internally mapping logical qubits that have been assigned to them by their home Computer onto physical qubits. In doing this, an ELU is required to reserve a user-specified percentage quota of its total physical qubits for use as ancilla in various computational process as described in section 4 below. An ELU object is also responsible for scheduling access to its limited number of concurrent internal operations and limited number of communication ports for entanglement generation.

- **Qubit class:** This is the workhorse class of simulator, making up the majority of the lines of code. For organizational and intuitive reasons, Qubit objects are used to represent both physical and logical qubits. Since these are two are two classes of object which interact with the rest of the simulator in significantly different ways, object and function members of a Qubit object frequently have different interpretations or even behaviors depending on whether the Qubit object is a physical qubit or a logical qubit. Qubit objects maintain a double which stores their fidelity. The interpretation and manipulation of this quantitiy is discussed in section 3 below. Qubits contain an integer which identifies the type, if any, of quantum error correcting code they are encoded in. Qubit objects maintain an array of pointers to Qubits; for logical qubits, pointers in this array lead to the lower-level constituent qubits that make up the logical qubit in the error correcting code that it is encoded in, while physical qubits simply leave this array null. Further information about the larger code structure the Qubit is a part of is stored in an array of integers specifying the code concatenation scheme, an integer indicating which level the Qubit resides in as part of the scheme, and a pointer to the Qubit, if any, that this Qubit is a constituent of. Instantiation of all physical qubits in a Computer occurs when the Computer's ELUs are constructed. Instantiation of a logical qubit recursively instantiates and links appropriate constituent qubits until final code concatenation level above the physical qubit level is reached. At this point, pointers to constituent physical Qubit objects are drawn from the target ELUs existing pool of physical qubits. Qubit objects maintain a double which stores their time of next availability. For a physical qubit, this stores the time at which the last operation involving the qubit was completed. For a logical qubit, this represents the latest time of next availability of all of the logical qubit's constituent qubits. Procedures for carrying out Pauli group, Hadamard, CNOT, and Toffoli operations are defined modularly on a per-code basis. Error correction and state preparation procedures are built up from these building blocks.

3. Modeling Errors and Error Correction

Each Qubit object maintains a double representing its fidelity. Broadly speaking, this number represents the probability that the actual state of the qubit is currently what it is supposed to be. In the absence of any error correction, this is the probability that every operation on the qubit up to the current time has been carried out perfectly without introducing any errors. Keeping track of a fidelity rather than a cumulative error probability simplifies computations by allowing the basic error process to be modeled by simply multiplying the fidelity of a qubit by the probability that an event introduced an error. Every physical qubit begins its existence with a fidelity of 1. Over the course of a simulation, three types of error events are applied to physical qubits.

- **Faulty gates:** Every actual physical operation carried out on a qubit has a user-defined operation-specific probability of being performed perfectly. Whenever the operation is simulated, the qubit's fidelity is multiplied by this probability.
- **Propogation of errors through multi-qubit operations:** If a qubit has undergone an error event and later is involved in a multi-qubit operations, it is possible for the error to propagate though the operation to affect any other qubit involved. To reflect this possibility, the fidelity of every qubit involved in a multi-qubit operation before the operation begins is applied to every other qubit involved in the operation as an error process. The net result computationally is for every qubit's individual fidelity to be replaced by the product of all involved qubit's fidelity. Of course, the error probability introduced by the gate itself must then also be applied to each qubit. Note that this process will lead to conservative estimates of fidelities since in reality some multi-qubit gates only propagate specific subsets of error types to specific subsets of target qubits [4]. Classically-controlled gates conditioned on the result of measurements also introduce error events. If the qubit is measured incorrectly, either because the measurement itself was faulty or the qubit was in an incorrect state prior to being measured, the classically-controlled gate will be applied (or not applied) inappropriately and will essentially constitute an error. As a conservative bound on the fidelity degradation this can cause, the fidelity of the measured qubit is applied as an error process to the other qubit.
- **Spontaneous decoherence:** Even qubits doing nothing at all are vulnerable to errors due to the possibility of unwanted interaction with the environment causing spontaneous decoherence [7]. This imperfect storage of quantum information results in memory errors which occur according to a Poisson process. To simulate this effect, if a qubit finishes an operation at time t_1 and is then idle for an interval before beginning another operation at time t_2, an error event of $\exp -\lambda(t_2 - t_1)$ is applied to the qubit prior the start of the second operation. λ is a user-defined time constant parameter reflecting performance characteristics of technology.

To combat these errors, of course, error correction procedures are used. Whenever a logical qubit undergoes an error correction process, the actual gates that make up the process as described above are first naively applied to the qubit. The fidelities of the constituent qubits are then extracted for processing (Fig. 11). If the constituent

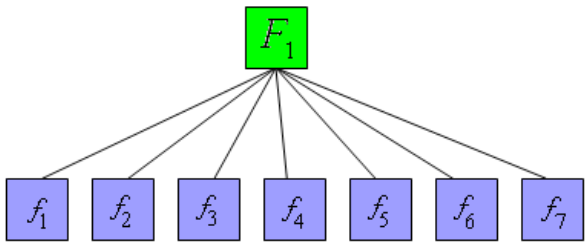

Figure 11: Fidelities of logical qubit and constituents before correction

qubits are themselves logical qubits, the fidelity extracted for each constituent is the product of the fidelity stored by the constituent and all of the fidelities of the constituent's constituents. As the workings of this process will reveal, this represents the probability that the logical constituent survived the last error correction round and has not been corrupted since that time. The fidelities of the logical qubits constituents are used to partition an event space $[0, 1]$ into regions corresponding to the probabilities of each relevant combination of error locations. For a code capable of correcting a single error such as the Steane code, these regions correspond to none of the constituents being in error, followed by each of the constituents being in error in isolation, followed by the constituents not being in a correctible state (Fig. 12). To stochastically simulate fidelity degradation due to possible corrective measures, a random event in the interval $[0, 1]$ is generated and a correction is applied corresponding to this events place in the event space. Following this, the partition is recalculated to determine the probability P_c that the logical qubit as a whole is in a correctible state (Fig. 13). The fidelities of constituent qubits are then reset to 1 to reflect the fact that the logical qubit has been projected back into a codeword state. The fidelity of the logical qubit is multiplied by the probability that it was correctible to reflect the chance that the qubit has been projected into the correct codeword state and has survived this round of error correction unscathed (Fig. 14).

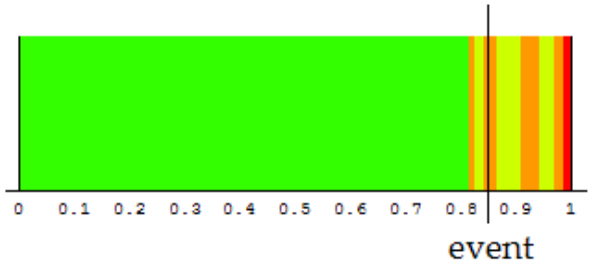

Figure 12: Partition of event space into error scenarios

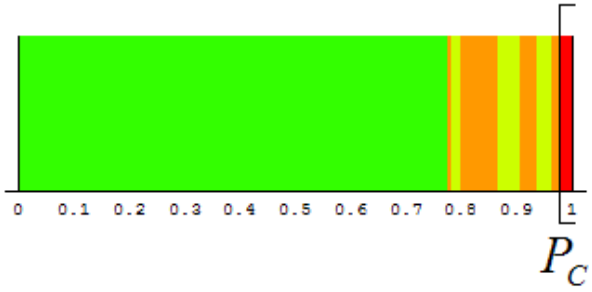

Figure 13: Recalculated partition with correctible probability

4. Scheduling Resource Access

In the ideal circuit description of a circuit being simulated, operations can appear to occur in parallel, both explicitly in the form of a vertical column of gates on separate qubits or implicitly in the form of simultaneous state preparations or error correction procedures. In reality, any such set of theoretically parallel operations will compete for access to finite pools of resources needed to actually carry out the operations.

There are four types of resources that are treated in this way.

- **Concurrent operations per ELU:** As a first order approximation to modeling the effects of limited physical operational hardware inside an ELU, an upper bound is placed on the number of concurrent physical operations that can be simultaneously underway in any given ELU.
- **Ancilla qubits:** Each ELU sets aside a quota of its total physical qubits for usage as ancilla qubits. Many processes associated with error correction procedures and remote gate operations require the temporary usage of ancilla as computational tools. Examples of such processes include those involving measuring an operator such as the general stabilizer code state preparation procedure and those derived from teleportation circuits such as the fault-tolerant Toffoli gate construction and the remote CNOT gate construction. Before any such process can be carried out, access to an appropriate number of ancilla must be reserved. Note that some processes involve role-swapping between data and ancilla qubits. In these cases, one-for-one exchanges occur between the set of physical qubits reserved for ancilla and those free for data usage.
- **Simultaneous entanglement generations per ELU:** Each ELU contains a limited number of communication ports used to carry out inter-ELU entanglement generation.
- **Concurrent global entanglement generations:** To accommodate modeling of different implementation strategies for entanglement generation, an upper bound is placed on the number of computer-wide entanglement generation processes which can be carried out simultaneously.

Contention for concurrent use of sparse resources is mediated by priority queues. The ResourceQueue class implements a heap-based priority queue [1] of Resource objects keyed on the time of next availability maintained by each Resource. At initialization, each Resource has its availability time set to zero. Whenever a Resource is

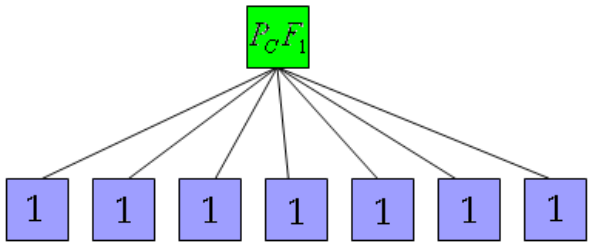

Figure 14: Fidelities of logical qubit and constituents after correction

needed by a process, the code responsible for simulating the process is required to pop a Resource off the top of the appropriate ResourceQueue. This requires processing of the remaining elements of the priority queue to preserve the heap property. Once all required resources have been reserved, the availability times of the required resources and the physical data qubits involved in the process are compared to determine the latest availability time. This represents the time at which the process is able to begin. Carrying out the process results in the availability times of all Resources and data qubits involved being advanced to the point in time equal to this beginning time plus the execution time of the process in question. All Resources are then reinserted into their respective ResourceQueue with this updated availability time and heap properties are appropriately reestablished. The net effect of this (pop, advance time, reinsert) process is to block out the time period on each Resource's schedule during which it was in use by the process. In this way, attempts to simultaneously use Resources of the same type are automatically parallelized up to the constraints imposed by the number of each resource. Beyond this point, further attempts at access are dynamically pushed back in time until previously used Resources become available again.

A heap-based implementation for priority queues was chosen because it provides the best combination of performance characteristics as the size of the queues scale upwards given the profile of the simulator's usage of the queues. This implementation allows resources to be popped off of the queue and reinserted into the queue in $O(\ln n)$ time, where n represents the size of the queue. Implementing the priority queues though linked lists would allow pops from the queue in $O(1)$ time at the expense of $O(n)$ time for reinsertions. Since every pop off a ResourceQueue will occur in a pair with a reinsertion, however, this tradeoff is undesirable.

Care must be taken whenever decisions in the simulator involving resource allocation are made based on the outcome of a procedure. For example, the results of a measurement of a qubit may be used to classically control the application of a gate on a another qubit. If the resources involved in the execution of this gate are not aged to reflect the time at which the decision-inducing procedure finished before they are used, information from the procedure will have essentially traveled backwards through time.

IV. Software Applications

In this section, three experiments conducted using Quipsim are presented. The experiments were selected and designed to demonstrate a range of Quipsim's capabilities while remaining accurate and relevant examples of the expected applications of the software.

Experiment 1 - Error Correction Strategy Evaluation

The frequency with which error correction is carried out on encoded logical qubits during the course of a computation represents a tradeoff between expected fidelity of the qubits at the conclusion of the computation and the time that it will take to reach this conclusion. Performing error correction with high frequency prevents a non-correctible number of errors from accumulating in between error correction cycles at the expense of the devotion of execution time to error correction rather than actually achieving a computation. Code concatenation schemes using more than one level of code complicate the situation considerably by allowing independent scheduling of error correction procedures at each code level. In the first part of this experiment, this tradeoff was explored by performing 128 X gates on an unencoded physical qubit as well as on a Steane logical qubit with no error correction performed, with error correction performed after every 4 X gates, and with error correction performed after every 64 X gates. The fidelity of the qubits and cumulative execution times over the course of the simulations

are shown for these four trials in Fig. 15. The exact values of all parameters used in this and later experiments can be found in Appendix A. In the fidelity plot on the left, the unencoded qubit undergoes a gradual decay in fidelity as the repeated application of X gates introduce potential errors that are never corrected. The uncorrected Steane encoded qubit undergoes significantly faster degradation in fidelity due to the fact that each logical X gate application results in seven physical X gates being transversally applied to the logical qubit's constituents. The sawtooth pattern to the error-correction-every-4 and error-correction-every-64 Steane qubits is characteristic of successful error correction. The two strategies result in virtually identical final fidelities. In the execution time plot on the left, a significant initial jump occurs between the unencoded qubit and the encoded qubits, reflecting the state preparation procedure the encoded qubits must undergo before the actual computation can begin. Final execution times reveal an order-of-magnitude lower time for the error-correction-every-64 Steane qubit than the error-correction-every-4 Steane qubit. The conclusion to be drawn by an experimentalist debating between error correction strategies is that is that for this set of parameters and circuit to be performed, error correction can be performed after every 64 or more gates to improve execution time with minimal loss of effectiveness in overall error correction capability.

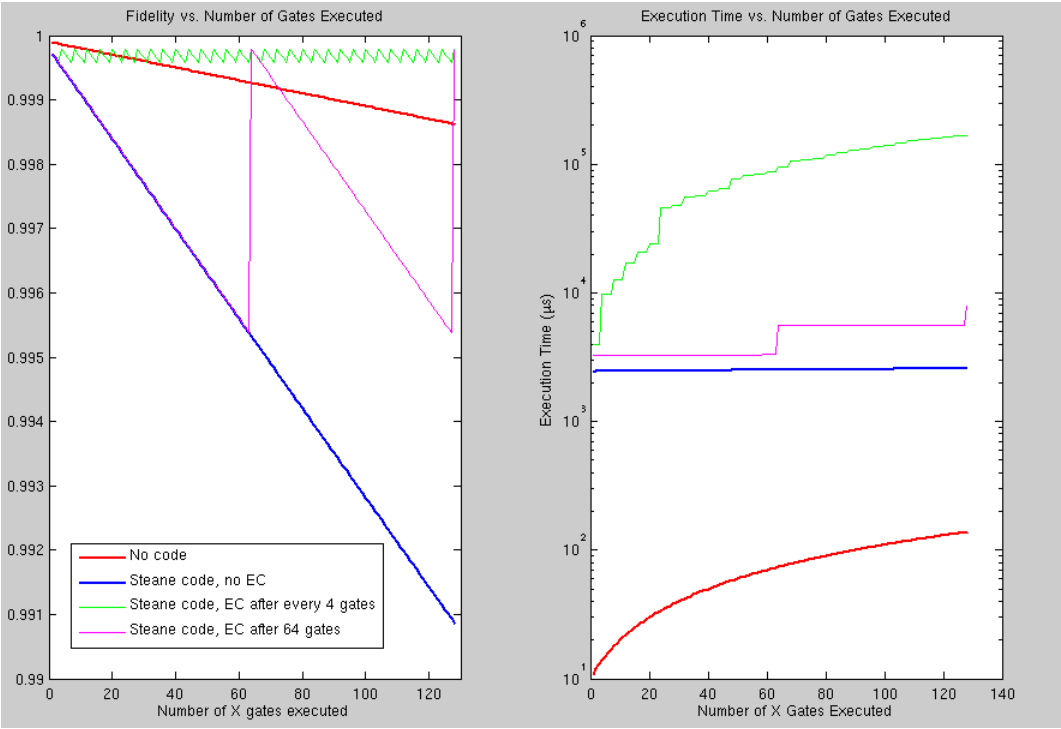

Figure 15: Error correction strategy tradeoffs

Carrying out error correction procedures involve incurring an inevitable initial fidelity degradation as the procedure is carried out via imperfect physical processes before the state purification properties of the error correction can occur. For some regions of the parameter space, the fidelity gain of the final purification may not offset this initial loss, causing the net result of a procedure to be an actual loss of fidelity. In the second part of this experiment, this effect was observed during evaluations of second level error correction scheduling for a two level Steane-Steane qubit undergoing the same computation as in the first part. Fig. 16 shows the final fidelity of the qubit with first level error correction occurring after every 4 gates and second level error correction occurring never, after every 4 gates, after every 16 gates, and after every 64 gates. The plot reveals that increasing the frequency of second level error correction actually causes a rapid drop in final fidelity. An experimentalist observing these results examine the output of the simulations to see that a combination of resource scarcity and spontaneous decoherence time constant in the computer being modeled led to the drops in fidelity whenever a second level error correction process attempted to prepare ancilla states to carry out the error correction, state preparation took so long that the data qubit had lost more fidelity while waiting due to spontaneous decoherence than the error correction procedure could repair.

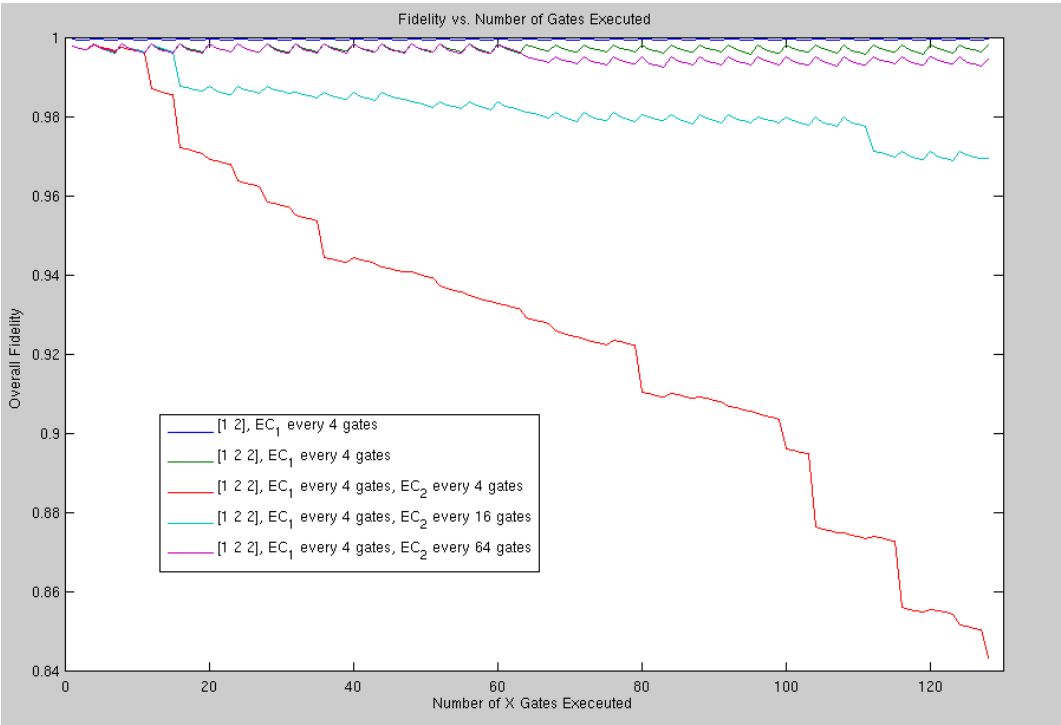

Figure 16: Non-viable error correction strategy

Experiment 2 - Dynamic Modeling of Communication Bottlenecks

As levels of code concatenation are layered on top of each other, the number of physical qubits needed to represent each logical qubit increases exponentially. For a given ELU size, this naturally reduces the number of logical qubits that can be fit onto a single ELU and forces more multi-qubit gates to assume remote forms requiring entanglement generation between ELUs. One of the major strengths of Quipsim is its ability to dynamically model the potential bottleneck that communication between ELUs can become under these circumstances. Fig. 17 illustrates this effect explicitly by showing the time needed to carry out a series of 10 CNOT gates between 10 pairs of qubits as well as the final overall fidelity of the ensemble of qubits for single level and two level Steane codes as ELU size is varied from the point where every qubit is contained on a single ELU to the point where the qubits are split evenly between two ELUs. In the computer being modeled, each ELU can carry out multiple concurrent local operations but only has one communication port for entanglement generation. This causes a shift from completely parallel execution of fast gates in the case of completely local operation to series execution of slower remote constructions as ELU size decreases and leads local operations that could be done in parallel shift to remote operations that must be done in series. Nearly four orders of magnitude of change in execution time result over the complete sweep of ELU size for the single level code. Non-monotonic behavior in this change is the result of stochastic modeling of successful entanglement generation as a Poisson arrival process; reduction in this behavior in the case of the two level code is the result of decreased variance in the joint distribution over a larger sample size of processes. An experimentalist observing these results could determine precisely how many physical qubits are needed per ELU to achieve a given performance level or possibly conclude that more communication ports should be added to ELUs to attempt to alleviate bottlenecks.

Experiment 3 - Gate Fidelity and Decoherence Time Dependencies of State Preparation

The standard fault-tolerant state preparation process for CSS codes such as the Steane code involves non-fault-tolerantly preparing four logical qubits then attempting to purify a single error free logical qubit out of the four in a post-selection process as described above. Simulations exploring the dependence of the expected time needed for this post-selection process to succeed on the fidelity of gate operations and the decoherence time constant produced the first real surprising results generated by Quipsim. Fig. 18 shows signs of these results.

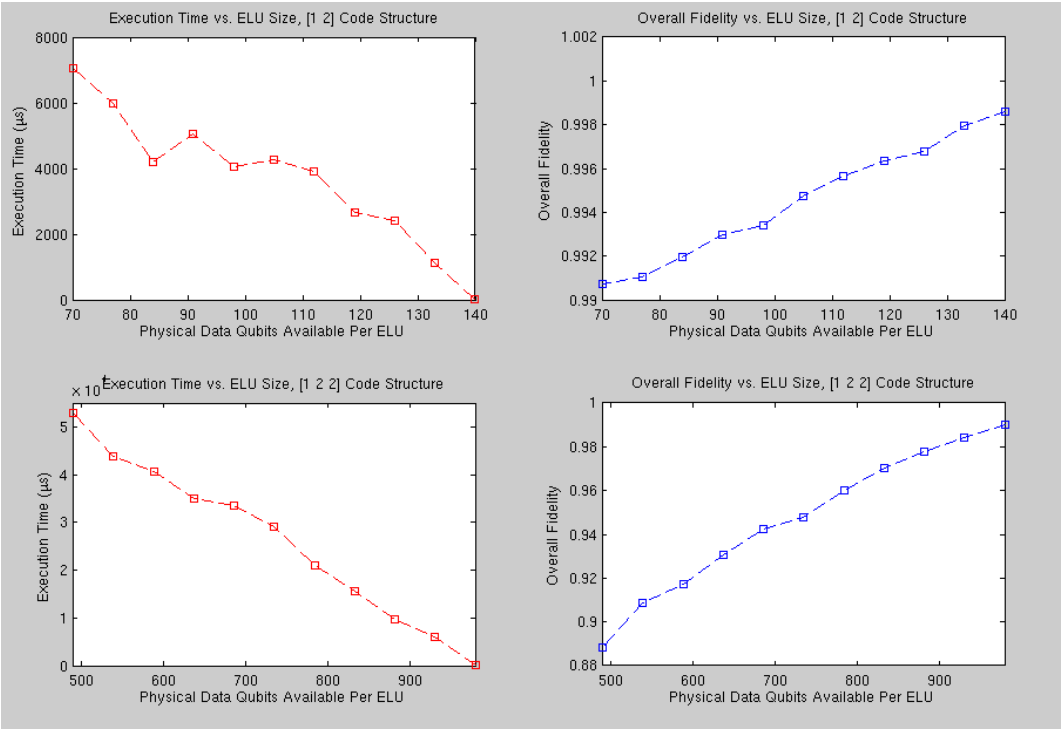

Figure 17: Dynamic modeling of communication bottlenecks

In the plots, blue crosses at each parameter value show the results of five identical simulations and reveal the extent of the variation that stochastic processes introduce, while the red lines connect the averages of each set of five simulations. In the top graph, a sense of the exponential increase in preparation time as the decoherence time constant parameter becomes larger can be gained, but the important feature of the experiment occurs just to the right of the end of the graph. Specifically, no results can be obtained for larger time constants because the probability of succeeding for each post-selection trial becomes to small for the simulator to report success in a reasonable run-time. The bottom graph shows this effect more explicitly as a four-fold increase in error probability leads to two orders of magnitude of increase in average preparation time. By revealing extremely sharp transitions in the difficulty of carrying out state preparation as parameters controlling the two mechanisms for fidelity degradation are varied, this experiment showed that Quipsim can guide experimentalists by producing tight, context-specific bounds on required performance characteristics of process technologies.

V. Future Work and Conclusion

Quipsim will be used to explore many interesting architectural ideas. One of the most obviously important such ideas is 'out-of-order' ancilla state preparation. Since ancilla qubits do not contain any information and are being prepared from scratch, we prefer to carry out this state preparation as close in time as possible to the point at which the ancilla will first interact with other resources. As soon as the qubit is prepared, it becomes data that is undergoing a spontaneous decoherence Poisson process. By delaying state preparation processes in situations where other resource constraints prevent immediately moving ahead with the prepared ancilla, the time interval during which decoherence can occur and by extension the entire computation's vulnerability to memory errors can be minimized. In the case of physical qubits, this can be done optimally: since physical state preparation time is an explicitly defined and constant parameter at the Computer level, state preparation can be scheduled to conclude precisely when all other bottlenecks limiting the ancilla's use resolve. In the case of logical qubits, the issue is more complicated. Since logical state preparation is a stochastically modeled process, logical state preparation time is not a well-defined constant but instead a distribution of times that is implicitly determined by the codes in use, the state preparation procedures chosen, the physical parameters of the various operations that make up these procedures, and the current state of resource availability. To approximate this

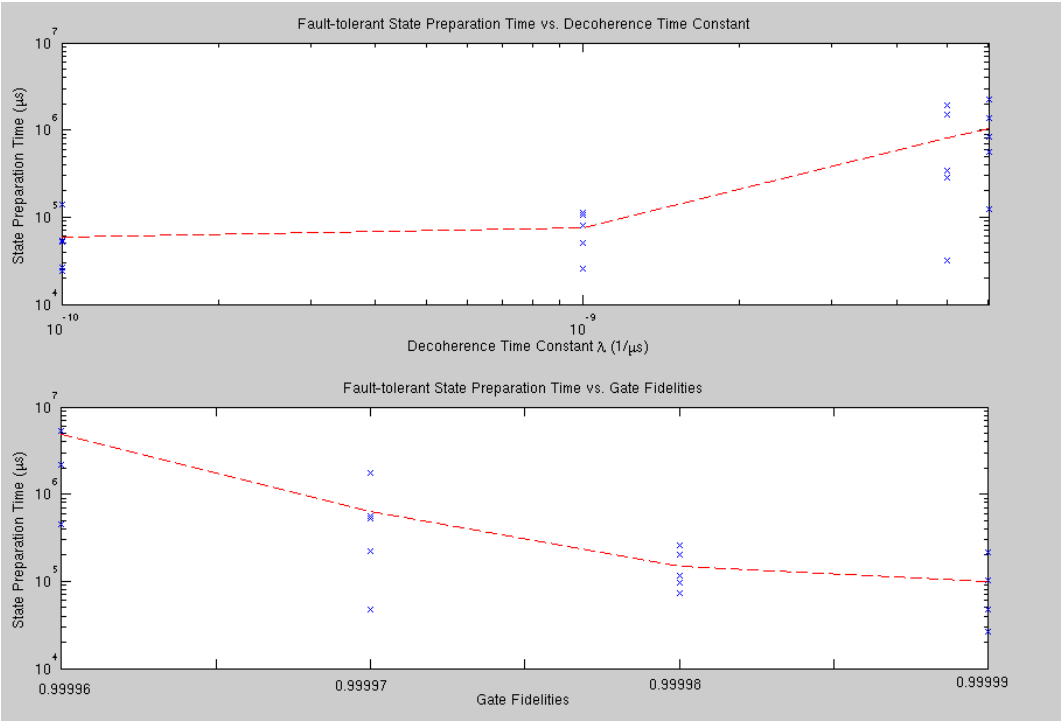

Figure 18: Exploration of physical parameter variations

distribution, Monte Carlo simulations of state preparation times given current resource availabilities need to be conducted in real time. Choosing a strategy for scheduling ancilla preparation based on this distribution becomes an architectural balancing act between the desire to ensure that a state has been successfully prepared when other bottlenecks resolve and the desire to limit the amount of time in advance of bottleneck resolution that this occurs.

In addition to causing unnecessary memory error exposure, premature state preparations also lead to inefficient resource usage by tying up qubits from ancilla allotment queues in an idle state in the potential presence of other concurrent needs for the ancilla. In an ideal scheduling scheme, other processes requiring ancilla could conceivably use the reserved resources and return them to the queue before the original reserving process is ready to use them itself. The central obstacle to implementing a system like this is that ancilla queue is by design entirely unaware of purpose for which ancilla are taken from it. Ancilla are checked out of the queue with no suggested or enforced return time. In particular, ancilla checked out by stochastically modeled post-selection processes could theoretically never return the ancilla to the queue. Naively scheduling resource allocation (that is, pushing operations back in time to enforced coordination points in logical order) avoids the prospect of unresolved needs at the cost of potentially suboptimal resource usage.

Other issue to be explored include component-wise error models for more accurate modeling of technology-specific error processes, arithmetic circuit design for use in the modular exponentiation stage of Shor's Algorithm, ELU location selection for ancilla state preparation in the fault-tolerant Toffoli construction, and modeling of technology-specific internal ELU operation via extension of the existing Resource/ResourceQueue framework.

Just as many interesting experiments in classical computer architecture can be carrying out by appropriately modifying existing simulators, the extensible design of the software I have made will allow a wide variety of questions in the nascent field of quantum computer architecture to be easily explored. By facilitating analysis of these types of issues, the simulator will provide guidance to the pioneering quantum computer architectures as well as a road map of technology requirements for experimentalists to work towards.

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2000.

[2] Daniel Gottesman. Stabilizer codes and quantum error correction. *quant-ph/9705052*, 1997.

[3] Daniel Gottesman. A theory of fault-tolerant quantum computation. *Physical Review A*, 57:127, 1998.

[4] Daniel Gottesman. Quantum error correction lectures at perimeter institute, 2007.

[5] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, 1988.

[6] Emanuel Knill and Raymond Laflamme. A theory of quantum error-correcting codes. *arXiv:quant-ph/9604034v1*, 1996.

[7] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, October 2000.

[8] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J.SCI.STATIST.COMPUT.*, 26:1484, 1997.

[9] Andrew M. Steane. Active stabilisation, quantum computation and quantum state synthesis. *arXiv:quant-ph/9611027v1*, 1996.

[10] Andrew M. Steane. Overhead and noise threshold of fault-tolerant quantum error correction. *arXiv:quant-ph/0207119v4*, 2003.

[11] Andrew M. Steane and Ben Ibinson. Fault-tolerant logical gate networks for css codes. *quant-ph/0311014v3*, 2005.

[12] Lieven M. K. Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S. Yannoni, Mark H. Sherwood, and Isaac L. Chuang. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 2001.

[13] Bei Zeng, Andrew Cross, and Isaac L. Chuang. Transversality versus universality for additive quantum codes. *arXiv:0706.1382v3*, 2007.

[14] Xinlan Zou, Debbie W. Leung, and Isaac L. Cuang. Methodology for quantum logic gate constructions. *arXiv:quant-ph/0002039v2*, 2000.

Appendix A

Parameters used in Experiment 1, Part 1

PHYSICAL_CNOT_ERROR: 0.99999
PHYSICAL_CNOT_TIME: 30
PHYSICAL_CZ_ERROR: 0.99999
PHYSICAL_CZ_TIME: 30
PHYSICAL_X_ERROR: 0.99999
PHYSICAL_X_TIME: 1
PHYSICAL_Y_ERROR: 0.99999
PHYSICAL_Y_TIME: 0.99999
PHYSICAL_Z_ERROR: 0.99999
PHYSICAL_Z_TIME: 1
PHYSICAL_H_ERROR: 0.99999
PHYSICAL_H_TIME: 1
PHYSICAL_TOFF_ERROR: 0.99999
PHYSICAL_TOFF_TIME: 100
PHYSICAL_MEAS_ERROR: 0.99999
PHYSICAL_MEAS_TIME: 200
PHYSICAL_STATE_PREP_TIME: 10
PHYSICAL_STATE_PREP_ERROR: 0.9999
LAMBDA: 1e-009
ENT_GEN_TIME: 1
ENT_GEN_CHANCE: 0.1
ENT_GEN_ERROR: 1
ENT_GEN_TIME: 1
MAX_ENT_GEN: 10
MAX_CONC_OPS: 100
COMM_PORTS: 1

Parameters used in Experiment 1, Part 2

PHYSICAL_CNOT_ERROR: 0.99999
PHYSICAL_CNOT_TIME: 30
PHYSICAL_CZ_ERROR: 0.99999
PHYSICAL_CZ_TIME: 30
PHYSICAL_X_ERROR: 0.99999
PHYSICAL_X_TIME: 1
PHYSICAL_Y_ERROR: 0.99999
PHYSICAL_Y_TIME: 0.99999
PHYSICAL_Z_ERROR: 0.99999
PHYSICAL_Z_TIME: 1
PHYSICAL_H_ERROR: 0.99999
PHYSICAL_H_TIME: 1
PHYSICAL_TOFF_ERROR: 0.99999
PHYSICAL_TOFF_TIME: 100
PHYSICAL_MEAS_ERROR: 0.99999
PHYSICAL_MEAS_TIME: 200
PHYSICAL_STATE_PREP_TIME: 10
PHYSICAL_STATE_PREP_ERROR: 0.9999
LAMBDA: 1e-009
ENT_GEN_TIME: 1
ENT_GEN_CHANCE: 0.1
ENT_GEN_ERROR: 1
ENT_GEN_TIME: 1
MAX_ENT_GEN: 10
MAX_CONC_OPS: 100
COMM_PORTS: 1

Parameters used in Experiment 2

PHYSICAL_CNOT_ERROR: 0.99999
PHYSICAL_CNOT_TIME: 30
PHYSICAL_CZ_ERROR: 0.99999
PHYSICAL_CZ_TIME: 30
PHYSICAL_X_ERROR: 0.99999
PHYSICAL_X_TIME: 1
PHYSICAL_Y_ERROR: 0.99999
PHYSICAL_Y_TIME: 0.99999
PHYSICAL_Z_ERROR: 0.99999
PHYSICAL_Z_TIME: 1
PHYSICAL_H_ERROR: 0.99999
PHYSICAL_H_TIME: 1
PHYSICAL_TOFF_ERROR: 0.99999
PHYSICAL_TOFF_TIME: 100
PHYSICAL_MEAS_ERROR: 0.99999
PHYSICAL_MEAS_TIME: 200
PHYSICAL_STATE_PREP_TIME: 10
PHYSICAL_STATE_PREP_ERROR: 0.99999
LAMBDA: 1e-009
ENT_GEN_TIME: 10
ENT_GEN_CHANCE: 0.1
ENT_GEN_ERROR: 0.99999
ENT_GEN_TIME: 10
MAX_ENT_GEN: 10
MAX_CONC_OPS: 100
COMM_PORTS: 1

Parameters used in Experiment 3

Code structure [1 2 2]
Physical qubits per logical qubit: 49
Number of ELUs: 20
Qubits per ELU: 400
Logical qubits per ELU: 2
Reserved ancilla qubits per ELU: 302

All gate fidelities were uniformly varied for the first part of the experiment. Lambda was uniformly varied for the second part of the experiment.
PHYSICAL_CNOT_ERROR: 0.99999
PHYSICAL_CNOT_TIME: 30
PHYSICAL_CZ_ERROR: 0.99999
PHYSICAL_CZ_TIME: 30
PHYSICAL_X_ERROR: 0.99999
PHYSICAL_X_TIME: 1
PHYSICAL_Y_ERROR: 0.99999
PHYSICAL_Y_TIME: 0.99999
PHYSICAL_Z_ERROR: 0.99999
PHYSICAL_Z_TIME: 1
PHYSICAL_H_ERROR: 0.99999
PHYSICAL_H_TIME: 1
PHYSICAL_TOFF_ERROR: 0.99999
PHYSICAL_TOFF_TIME: 100
PHYSICAL_MEAS_ERROR: 0.99999
PHYSICAL_MEAS_TIME: 200
PHYSICAL_STATE_PREP_TIME: 10
PHYSICAL_STATE_PREP_ERROR: 0.99999
LAMBDA: 5e-009
ENT_GEN_TIME: 10
ENT_GEN_CHANCE: 0.1
ENT_GEN_ERROR: 0.99999
ENT_GEN_TIME: 10
MAX_ENT_GEN: 10
MAX_CONC_OPS: 100
COMM_PORTS: 1