Study of $[[4,2,2]]$-Concatenated Toric Code for a Scalable Circuit-QED Architecture

Author: Tobias Hölzer

Supervisor: Prof. Dr. Barbara M. Terhal

Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Physics

Faculty of Mathematics, Informatics and Physics

September 2013
Quantum Error Correction Sonnet

We cannot clone, perforce; instead, we split
Coherence to protect it from that wrong
That would destroy our valued quantum bit
And make our computation take too long.

Correct a flip and phase - that will suffice.
If in our code another error’s bred,
We simply measure it, then God plays dice,
Collapsing it to X or Y or Zed.

We start with noisy seven, nine, or five
And end with perfect one. To better spot
Those flaws we must avoid, we first must strive
To find which ones commute and which do not.

With group and eigenstate, we’ve learned to fix
Your quantum errors with our quantum tricks.

BY DANIEL GOTTESMAN, 1999
1 Introduction

This chapter provides a short introduction to the topic and leads to the subject of this thesis.

1.1 The Problem

A quantum computer is a device performing information processing on states using quantum mechanical phenomena. A quantum algorithm can make use of these phenomena such as entanglement or superposition of states. If the quantum computer is of appropriate size, it can outperform any classical computer regarding certain algorithms. Among those are integer factorization and the simulation of quantum many body systems[12].

However, a quantum system is unstable. Like classical computers, quantum computers are imperfect. Storing data, operating gates and transmitting information is faulty by default. While a classical bit in a hard drive is hard to corrupt, a quantum bit (qubit) is highly unstable. Even at very low temperatures, there are uncontrollable interactions with the environment. Therefore, coherent qubit states rapidly decohere. There is a variety of schemes how to encode information to stabilize and preserve the data using fault-tolerant recovery protocols. All of them work with the replacement of physical qubits by encoded qubits incurring overhead in space and time.

So to put the problem in a nutshell, quantum computers outperform classical computers regarding certain algorithms, but are imperfect as well. What we need are stable, locally strongly coupled qubits and quantum error correcting codes that allow recovery.

1.2 The Approach

Is there anything we can do? A promising class of quantum error correcting codes are the surface codes proposed by Kitaev [9, 10], where qubits are arranged on a two-dimensional lattice. The properties of the lattice are determinating for the encoded information. If the error rate per qubit is below the accuracy threshold, error recovery succeeds with a probability that rapidly approaches one for an increasing lattice size. The numerical value $p_c$ of the accuracy threshold is of considerable interest, since it characterizes how reliably quantum hardware must perform in order for a quantum memory to be robust [17].
Chapter 1. Introduction

And here, practicability and especially qubit implementation should be considered. A promising physical qubit is the Transmon, a superconducting qubit consisting of a Josephson junction shunted by a large capacitor [6]. Recent approaches propose the coupling of superconducting qubits like the Transmon to three-dimensional cavities. The cavities protect the qubits from outside world influences and therefore dramatically increase coherence times. A physical toolbox for error correction is also available [13].

The approach made by B. Terhal is to use the capability of stronger, better controlled local versus weaker, noisier coupling between designated sets of qubits which couple more remotely to other qubits [15]. A reasonable quantum computer architecture is a combination of superconducting qubits like Transmons coupled to 3D cavities and the surface/toric error recovery code. This results in a \([4,2,2]\)-concatenated toric code for a scalable circuit-QED architecture. In this thesis the code and especially the accuracy threshold in various regimes for perfect and noisy error correction is studied.
2 The Toric Code Error Correction

This chapter provides brief background information about the stabilizer formalism, toric code error correction and the accuracy threshold. At the end, the $[[4,2,2]]$-concatenated toric code is introduced.

2.1 Operators and the Stabilizer Formalism

We consider an $L \times L$ toric lattice$^1$. A number of $k = 2$ logical qubits$^2$ is encoded in $N = 2L^2$ qubits that are placed on each edge $e_i$. The distance$^3$ of the toric code is $d = L$, making it a $[[N,k,d]] = [[2L^2, 2, L]]$ code.

![Toric Lattice Diagram]

Figure 2.1.1 – The toric lattice

This system can encode logical qubits of well protected quantum information if the error rate is below the threshold $p_c$. A number of $2L^2$ ancilla qubits are placed on each site $\sigma_i$ and plaquette $\pi_i$ of the lattice. Each site and plaquette is associated with a certain check operator so that the ancilla qubits receive the parity of the four surrounding qubits as seen in figure 2.1.2.

---

$^1$While a planar surface code is also possible and even more realistic, a toric lattice simplifies the numerical implementation.

$^2$Later, we will consider the second logical qubit to be a gauge qubit.

$^3$The distance $d$ is the minimum number of qubits a logical operator acts nontrivial on. See figure 2.1.4.
Chapter 2. The Toric Code Error Correction

We can describe every operation on the qubit states as a linear combination of the Pauli matrices and the $2 \times 2$ identity times $\pm i, \pm 1$, for which we will use the following notation:

\[
I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.
\] (2.1.1)

The check operator at site $\sigma_j$ acts non-trivially on the four surrounding qubits on the edges $\epsilon_i \in \sigma_j$. Let $X_i$ be the operator that acts on $\epsilon_i$, then the site operator becomes the tensor product

\[
X_{\sigma_j} = \left( \otimes_{\epsilon_i \in \sigma_j} X_i \right) \left( \otimes_{\epsilon_i \notin \sigma_j} I_i \right).
\] (2.1.2)

The check operator at plaquette $\pi_j$ acts non-trivially on the four surrounding qubits on the edges $\epsilon_i \in \pi_j$. The site operator is the tensor product

\[
Z_{\pi_j} = \left( \otimes_{\epsilon_i \in \pi_j} Z_i \right) \left( \otimes_{\epsilon_i \notin \pi_j} I_i \right).
\] (2.1.3)

The operators can be observed in figure 2.1.2. The plaquette and site operators have eigenvalues $\pm 1$ and independently commute with each other. So with the commutation operator being

\[
[A, B] = AB - BA,
\] (2.1.4)

the relation between the operators becomes

\[
[X_{\pi_j}, X_{\pi_k}] = [Z_{\sigma_j}, X_{\pi_k}] = [Z_{\sigma_j}, Z_{\sigma_k}] = 0, \ \forall j, k \in L \times L.
\] (2.1.5)

The plaquette and site operators altogether generate an Abelian group

\[
S = \{ I, X_{\pi_j}, Z_{\sigma_j}, j \in L \times L \}
\] (2.1.6)
which we call Stabilizer group. We define the +1 eigenspace of the group S as the Codespace

\[ C = \left\{ |\psi\rangle : X_{\pi j}|\psi\rangle = Z_{\sigma j}|\psi\rangle = |\psi\rangle, \ \forall j \in L \times L \right\} \tag{2.1.7} \]
on which we will operate. Because of the periodic boundaries, the product of all site or plaquette operators is the identity.

We consider a Codeword \(|\varphi\rangle \in C\) to be in the Codespace. Other products of \(X_i\) or \(Z_i\) are imaginable to act on the lattice as well. Trivial loops are products of plaquette or site operators

\[ X_{\text{trivial}} = \bigotimes_j X_{\pi j} X_{\gamma j} \leftrightarrow X_{\text{trivial}}|\varphi\rangle \in C, \ \forall \gamma j \in \{0, 1\} \tag{2.1.8} \]

which commute with \(S\), act trivially on the codeword and are therefore harmless.

There are operators which also commute with all elements in \(S\), but are not of the form (2.1.8). These are the logical operators

\[ X_{\text{nontrivial}} = \bigotimes_i X_i \leftrightarrow X_{\text{nontrivial}}|\varphi\rangle \notin C \tag{2.1.9} \]
represented by nontrivial loops on the lattice (as seen in figure 2.1.3) which map a codeword out of the Codespace.

Then, there are products of Pauli’s which do not commute with all elements in \(S\). These are the errors. Such an operator is a nontrivial loop on the torus. For the toric code, there are 4 of these, which we call logical operators:

\[ X_1, X_2, Z_1, Z_2. \tag{2.1.10} \]

If a logical operator acts on the lattice, the data is corrupted and is regarded as a logical error [14, 17, 12]. The logical operators can be seen in figure 2.1.4.
2.2 Errors and Correction

Errors We imagine only Pauli errors to occur, which is called depolarizing noise model. By definition, Pauli errors anti-commute with at least one element of the Stabilizer group. $X$ errors anti-commute with some plaquette operators, $Z$ errors anti-commute with site operators and $Y$ errors are products of $X$ and $Z$. An error $E$ somewhere on the lattice anti-commutes with some site or plaquette operators that lays next to it. The place where a parity check operator has the eigenvalue $-1$ is called a defect. So by measur-
Chapter 2. The Toric Code Error Correction

ing the site and plaquette operators, which are the generators of \( S \), we get information about what errors occurred and are able to correct them.[14]

**Correction** Imagine only \( X \) errors to occur. \( Z \) errors are independently dealt with in the dual lattice and \( Y \) errors are \( X, Z \) products. To diagnose the errors, the code’s local check operators are measured at each lattice site. This provides isolated site defects at the ends of the chain, which we call syndrome \( \text{syn} \). But those syndromes are ambiguous. Since a measurement only provides the boundary \( \partial E \) of an error chain \( E \), any error chain \( E' \) with the same boundary results in an equal syndrome

\[
\text{syn}(E) = \text{syn}(E'),
\]

as seen in figure 2.2.1.

![Defects](image)

**Figure 2.2.1** – Defects

However, we are able to use (2.2.1) to recover from an error \( E \). We choose any minimum distance recovery chain \( E' \) with

\[
\partial E' = \partial E,
\]

then apply \( X \)’s to all the qubits on \( E' \). Since (2.2.2),

\[
D = EE'
\]

is always a cycle with \( \partial D = 0 \).

**Good Decoding** If \( D \) is a trivial loop on the lattice, the error correction succeeded as in figure 2.2.2. Applying \( D \) equals applying a product of check operators which act trivially on the Codespace. Therefore, the encoded qubits are well protected.
Failed Decoding  If $D$ is a nontrivial loop on the lattice, as seen in figure 2.2.3, then $D$ commutes with all elements of $S$, but is not in $S$:

\[
[D, S] \neq 0,  \quad D \notin S.  \tag{2.2.4}
\]

The recovery fails and the encoded information suffers an error.
2.3 Noisy Error Correction

The model described above only applies to an unrealistic scenario, where the measurement outcomes of the check operators are perfect. But these measurements are qubit measurements and are themselves degraded by noise. We will call the error probability for the syndromes per time step $q$.

(a) Defects in spacetime: Red dots are vertices, where the syndrome changed in time.

(b) Spacetime defect matching

Figure 2.3.1 – 3D spacetime, taken from [4].

(a) Each row represents the syndrome at a particular time. Qubits reside on plaquettes, and 2 qubit check operators are measured at each vertical link. Nontrivial syndromes are shaded Grey.

(b) Links where errors occurred are darkly shaded, nontrivial syndromes lightly. Horizontal links indicate qubit flips, vertical links indicate faulty syndrome measurements. Vertical links both lightly and darkly shaded are locations where a nontrivial syndrome was found erroneously.

Figure 2.3.2 – 1D cross section through spacetime, taken from [4].

The solution is to repeat the syndrome measurement $L$ times. Assembling a measurement history, we decode in a three-dimensional spacetime. Vertices in spacetime are considered defects if the syndrome of a check operator changes its value in time as seen in figure 2.3.1. Again, we try to find a minimum distance between defects.
Consider a 1D cross-section through the spacetime lattice. Qubits are on horizontal links, vertical links represent check syndrome outcomes as seen in figure 2.3.2. To decode, we need to bring pairs of defects together trivially and apply the appropriate operators to all the qubits that lie on the connection lines in the space dimensions. [4, 16]

2.4 The Accuracy Threshold

The critical value \( p_c \) is called the accuracy threshold and is of considerable interest. If the error rate per qubit \( p \) is below \( p_c \), error recovery succeeds with a probability that rapidly approaches one for an increasing lattice size. However, if \( p \) is above \( p_c \), the failure probability approaches a nonzero constant for an increasing lattice size as seen in figure 2.4.1. The accuracy threshold is the determining value characterizing how reliably quantum hardware must perform in order for a quantum memory to be robust [17].

![Figure 2.4.1](image-url) – An overview of the failure rate for noise free concatenated toric code. The data points are generated with \( 10^3 \) iterations.
2.5 The Transmon

The Transmon is a superconducting qubit consisting of a Josephson junction shunted by a large capacitor, as seen in figure 2.5.1.

The Josephson junction generates anharmonicity so that the eigenenergies of the Transmon Hamiltonian are well separated. For a specific regime \( \frac{E_J}{E_C} \approx 30 \), there is no dispersion through outside fields (see figure 2.5.2).

Figure 2.5.1 – The Transmon: b) Optical micrography and a) schematic circuit diagram. The use of two Josephson junctions allows tuning of the effective Josephson energy via the external magnetic flux penetrating the superconducting loop. Graphic taken from [8].

The Josephson junction generates anharmonicity so that the eigenenergies of the Transmon Hamiltonian are well separated. For a specific regime \( \frac{E_J}{E_C} \approx 30 \), there is no dispersion through outside fields (see figure 2.5.2).

Figure 2.5.2 – Eigenenergies \( E_m \) (first three levels, \( m = 0, 1, 2 \)) of the CPB Hamiltonian (1) as a function of the effective offset charge \( n_g \) for different ratios \( \frac{E_J}{E_C} \). All energies are given in units of the transition energy \( E_{01} \) (evaluated at the degeneracy point \( n_g = 1/2 \)). The zero point of energy is chosen as the bottom of the \( m = 0 \) level. The sequence of plots highlights the exponentially increasing flatness of energy levels and the slow loss of anharmonicity as \( \frac{E_J}{E_C} \) is increased. The third graphic from left is the Transmon regime. Graphic taken from [8].
Chapter 2. The Toric Code Error Correction

This allows us to target energy levels separately and use them as ground and excited state. The planar circuit-QED architecture of such a qubit works with a superconducting co-planar waveguide resonator, as seen in figure 2.5.3.

![Figure 2.5.3](image)

Figure 2.5.3 – Schematic layout of the circuit QED architecture. Two superconducting Transmons are fabricated inside a superconducting co-planar waveguide resonator. Graphic taken from [1].

A Transmon qubit is coupled to the waveguide resonator which enables certain micro wavelengths to exist. This is how the desired states can be targeted. Microwaves are used for preparation, processing and outreading of the qubit. An electromagnetic single shot readout signal acquires a phase shift depending on the state of the qubit [6]. This signal determines the Transmon’s state and even allows parity check measurements if we couple more than one Transmon to the waveguide resonator [5]. However, an architecture with more than three Transmons on a co-planar waveguide turns out to be very hard to implement. A recent approach is the coupling of Transmons to 3D cavities, as seen in figure 2.5.4.

![Figure 2.5.4](image)

Figure 2.5.4 – Transmons coupled to 3D cavities, taken from[13].

The cavities protect the qubits from outside world influences and therefore dramatically increase coherence times. Even a single qubit targeting via Spin Echo Readout
is possible so this scheme is a promising candidate to incorporate into the toric error recovery code.

2.6 The [[4,2,2]]-Concatenated Toric Code

So why not implement an architecture, that uses the capability of stronger, better controlled local versus weaker, noisier coupling between designated sets of qubits which couple more remotely to other qubits?[15] A reasonable architecture is a combination of superconducting qubits like Transmons coupled to 3D cavities plus the surface/toric error recovery code. The idea is to use an $L \times L$ lattice, in which every qubit is replaced by a 3D cavity with 4 superconducting qubits as seen in figure 2.6.1. 4-qubit $X$ and $Z$ parity check operators are performed with the qubits in the cavities (we will call them square parity checks), and 8-qubit $X$ and $Z$ parity checks are performed with 2 qubits from each of the 4 surrounding cavities (we will call them octagon parity checks). This results in a [[4,2,2]]-concatenated toric code that is represented by a square-octagon lattice (figure 2.6.2).

The Question is how much we make things worse with weight-8 parity checks instead of weight-4 parity measurements in the toric code. For noise free error correction, the concatenated toric code might be slightly worse. Interesting is the noisy error correction where the error probability of the square checks $q_S$ differs from the octagon’s $q_O$. We study the accuracy threshold in various regimes for perfect and noisy error correction.
Chapter 2. The Toric Code Error Correction

Figure 2.6.2 – Square-octagon lattice. The qubits got replaced by 4 Transmons in 3D cavities. $X$ and $Z$ square parity checks take place inside the cavities where they measure the 4-qubit’s parities. $X$ octagon parity checks take place at the former lattice sites. $Z$ octagon parity checks take place at the former lattice plaquettes. Octagons check the parities of the 8 neighboring qubits.
3 Numerics

In this chapter, we develop a procedure to find the accuracy threshold $p_c$ for noise free and noisy error correction in each case for the toric and the concatenated toric code.

3.1 Assumptions

To simulate the error recovery process, we need to make some assumptions about the nature of the noise and how the computer operates. These assumptions are mostly transcribed from [4].

- **Constant error rate.** We assume that the strength of the noise is independent of the number of qubits in the computer. If the noise increases as we add qubits, then we cannot reduce the error rate to an arbitrarily low value by increasing the size of the code block.

- **Weakly correlated errors.** Errors must not be too strongly correlated, either in space or in time. In particular, fault-tolerant procedures fail if errors act simultaneously on many qubits in the same code block.

- **Parallel operation.** We need to be able to perform many quantum gates in a single time step. Errors occur at a constant rate per unit time, and we are to control these errors through information processing. We could never keep up with the accumulating errors except by doing processing in different parts of the computer at the same time.

- **Reusable memory.** Errors introduce entropy into the computer, which must be flushed out by the error recovery procedure. Quantum processing transfers the entropy from the qubits that encode the protected data to “ancilla” qubits that can be discarded. Thus fresh ancilla qubits must be continually available. The ability to erase (or replace) the ancilla quickly is an essential hardware requirement.

- **Fast measurements.** It is helpful to assume that a qubit can be measured as quickly as a quantum gate can be executed. For some implementations, this may not be a realistic assumption — measurement requires the amplification of a microscopic quantum effect to a macroscopic signal, which may take a while.

- **Fast and accurate classical processing.** If classical processing is faster and more accurate than quantum processing, then it is beneficial to substitute classical processing for quantum processing when possible. In particular, if the syndrome is measured, then a classical computation can be executed to determine how recovery should proceed.
3.2 The General Procedure

3.2.1 Noise Free Error Correction

Toric Code

1. We produce an \( L \times L \) lattice with \( 2L^2 \) qubits and \( 2L^2 \) ancilla qubits that represent the syndrome check outcomes. Let the starting state of the lattice \( |\varphi\rangle_0 \) be somewhere in the Codespace \( C \):

\[
|\varphi\rangle = |\varphi\rangle_0.
\]  
(3.2.1)

2. At each qubit, we generate an \( X \) error with the probability \( p \). This results in an error \( E \) of the lattice. For convenience, we are only considering \( X \) errors, since \( Z \) errors are dealt with independently and \( Y \) errors are products of \( X \) and \( Z \). The lattice state is now

\[
|\varphi\rangle = E|\varphi\rangle_0.
\]  
(3.2.2)

3. We do syndrome measurements at each site. This results in a syndrome \( syn(E) \).

4. Then, we calculate the distance from every defect to all the others. Afterwards, we apply a minimum weight matching algorithm \( MWM() \) to \( syn(E) \) that minimizes the overall distance between pairs of defects. This results in an inferred (or guessed) error

\[
E_G = MWM(syn(E)).
\]  
(3.2.3)

We should keep

\[
syn(E_G) = syn(E)
\]  
(3.2.4)
in mind.

5. We apply error correction to every matched pair of defects. This means applying \( X \)'s to all the qubits on the shortest path from every defect to the matched partner. This results in the lattice state

\[
|\varphi\rangle = E_G E |\varphi\rangle_0 = P |\varphi\rangle_0.
\]  
(3.2.5)

6. We check if the error correction procedure was successful by testing whether

\[
P = E_G E
\]  
(3.2.6)

commutes with \( Z_1 \) (see (2.1.10)). This works because the overlap between a nontrivial action on qubits \( X_1 \) and \( Z_1 \) is odd, while the overlap for nontrivial loops \( P \neq X_1 \) with \( Z_1 \) is even.

\[
[P, Z_1] = 0 \iff P \neq X_1 \iff \text{Correction succeeded}
\]

\[
\neq 0 \iff P = X_1 \iff \text{Correction failed}
\]

(3.2.7)

We are only testing the commutation relation between \( P \) and \( Z_1 \), because we defined the second encoded qubit in the lattice to be gauge data and do not deal with
Z errors. Otherwise, \( P \) must commute with all logical operators \( X_1, Z_1, X_2 \) and \( Z_2 \) in order to successfully protect the information. However, this procedure does not alter the accuracy threshold.

7. We then repeat step 1-6 \( N \) times for constant lattice sizes \( L \) and error probabilities \( p \), count the number of failed error recovery attempts \( N_{\text{fail}} \) and divide by \( N \) to get the failure rate \( f_L(p) \). In several steps, we alter \( L \) and \( p \) to find the accuracy threshold \( p_c \). \( p_c \) is the \( p \) at which all lattices have the same failure rate.

**Concatenated Toric Code** This procedure is equal for the toric and the concatenated toric code. However,

- there are \( 4 \times 2L^2 \) qubits on an \( L \times L \) square-octagon lattice and
- the logical operators look slightly different, as seen in figure 3.2.1.

\[
\begin{align*}
|\psi\rangle_0 &= |\psi\rangle,
\end{align*}
\]

\( (3.2.8) \)

**3.2.2 Noisy Error Correction**

**Toric Code**

1. We produce an \( L \times L \) lattice with \( 2L^2 \) qubits and \( 2L^2 \) ancilla qubits that represent the syndrome check outcomes. The error free \( T = 0 \) state of the lattice \( |\psi\rangle_0 \) shall be in the Codespace \( C^r \):

\[
|\varphi\rangle = |\psi\rangle_0.
\]

\( (3.2.8) \)

2. We prepare an \( L \times L \times (T + 1) \) array that stores the defect history \( H \). The third dimension is the amount of time steps \( T = L \) errors are propagated in the lattice. To propagate \( T = L \) times is a reasonable calculation as seen in figure 3.2.2.
Chapter 3. Numerics

Figure 3.2.2 – The failure rate for L = 8 and different propagation times T. We see that there is an upper limit for the failure rate. More time steps generate more errors, but also generate more information about the lattice state.

3. At each qubit, we generate an X error with the probability p. This results in an error $E_t$ of the lattice.

4. We do syndrome measurements at each site. With a probability q, we flip the result of every parity measurement outcome. This results in a syndrome $syn_q(E_t)$. If the syndrome changed at a point in spacetime (in the time dimension) we consider it a defect. The spacetime coordinates of each defect are stored in the defect history $H$.

5. We repeat step 3 & 4 $T$ times. The lattice state is now

$$|\varphi\rangle = E|\varphi\rangle_0 = \otimes_{t=1}^{T} E_t |\varphi\rangle_0.$$  \hspace{1cm} (3.2.9)

6. Then, we calculate the distance from every defect to all the others. This should be done with respect to different edge weights, if $q \neq p$. As shown in [4], the edges in space should be weighted with

$$w_s = \log \left( \frac{1 - p}{p} \right)$$  \hspace{1cm} (3.2.10)

and the edges in time with

$$w_t = \log \left( \frac{1 - q}{q} \right),$$  \hspace{1cm} (3.2.11)

respectively. Then, we apply a minimum weight matching algorithm MWM($H$) to $H$ that minimizes the overall distance between pairs of spacetime defects. This results in an inferred error $E'_G = \text{MWM}(H)$.

7. We apply error correction to every matched pair of defects whose connecting path has space-components. This results in the lattice state

$$|\varphi\rangle = E'_G E|\varphi\rangle_0 = P'|\varphi\rangle_0.$$  \hspace{1cm} (3.2.12)
In reality, we would project all the matchings paths of $H$ onto the final boundary (If an error occurred twice on a qubit, we do not need to correct anything). The projection ensures to only correct the remaining errors.

8. We do one round of error correction with perfectly executed syndrome measurements. This should be discussed further. The error syndrome is faulty:

$$\text{syn}(E_G) \neq \text{syn}(E), \text{ for most } E_G, E.$$  \hspace{1cm} (3.2.13)

Therefore, some residual errors $E_r$ can be left in the lattice and $P'$ does not map $|\varphi\rangle$ onto the Codespace:

$$|\varphi\rangle = E_G' E |\varphi\rangle_0 = P' |\varphi\rangle_0 = E_t |\varphi\rangle_0.$$  \hspace{1cm} (3.2.14)

So we need to do a last step of perfect correction in order to correct $E_r$ perfectly. This is applying step 3, 4 and 5 from the noise free error correction to the lattice. To formalize this, we do a perfect measurement of $\text{syn}(E_r)$, compute the Minimum Weight Matching algorithm $\text{MWM}(\text{syn}(E_r))$ to infer an error $E_G = \text{MWM}(\text{syn}(E_r))$ and apply error correction to the qubits on the paths. This results in a lattice state

$$|\varphi\rangle = E_G E_r |\varphi\rangle_0 = P |\varphi\rangle_0$$  \hspace{1cm} (3.2.15)

which is finally an element of the Codespace again$^1$.

9. We need to check if the error correction procedure was successful by testing whether

$$P = E_G E_r = E_G P' = E_G E'_G E$$  \hspace{1cm} (3.2.16)

commutes with $Z_1$. As before$^2$:

$$[P, Z_1] \begin{cases} = 0 \iff P \neq X_1 \iff \text{Correction succeeded} \\ \neq 0 \iff P = X_1 \iff \text{Correction failed} \end{cases}.$$  \hspace{1cm} (3.2.17)

10. We then repeat step 1-6 $N$ times for constant lattice sizes $L$ and error probabilities $p$, count the number of failed error recovery attempts $N_{\text{fail}}$ and divide by $N$ to get the failure rate $f_L(p)$. We alter $L$ and $p$ to find $p_c$. The accuracy threshold $p_c$ is that $p$, at which all lattices have the same failure rate.

**Concatenated Toric Code** In general, the procedure is equal for the toric and the concatenated toric code, except for the following modifications:

- This time there are $4 \times 2L^2$ qubits on an $L \times L$ square-octagon lattice.
- The logical operators look slightly different, as seen in figure 3.2.1.

$^1$This procedure does not seem to represent the fact, that there are only imperfect syndrome measurements. Experimentally, we had to do a physical outreading of the data and had to prepare the state again afterwards.

$^2$With a gauge qubit and only considering $X$ errors.
Chapter 3. Numerics

- In step 4, the parity check outcome for square checks is flipped with a probability $q\Box$, while the octagon checks are flipped with a probability $q\bigcirc$.

3.3 The Simulation

Here, the programming implementation for every nontrivial step from the general procedure is described. We may keep in mind that any realization which allows multi threading\(^3\) should be always favored.

The programming code can be found here [3] and is based on the work of Martin Suchara, UC Berkeley [2].

3.3.1 Noise Free Error Correction

Toric Code

1. To produce an $L \times L$ lattice, we use a $[2L][2L]$ array, where each parity checker and each qubit has a place in the array.

2. The generation of errors is done with srand(). Note that one should always use a fixed random numbers seed while debugging.

3. To do syndrome measurements at each site, we need to be careful with adding and subtracting array indices on a torus.

4. Calculating the distance $D$ between two qubits $Q_{xy}$, $Q'_{xy}$ on a toric square lattice is done with a Manhattan metric. But we have to pay attention to the torus. Therefore:

   $D = \min(|x_Q - x_{Q'}|, L - |x_Q - x_{Q'}|) + \min(|y_Q - y_{Q'}|, L - |y_Q - y_{Q'}|). \quad (3.3.1)$

5. We use the Blossom-V minimum weight matching algorithm by Vladimier Kolmogorov [11].

6. Error correction is done by independently going the straight paths in $x$ and $y$ direction.

Concatenated Toric Code

1. To produce an $L \times L$ lattice, a lot of schemes are possible. For convenience, we use a $[2L][2L][5]$ array, where each parity checker and each qubit has a place in the array. The $[1][0]$ layer is regarded the syndrome layer while $[1][1][1][1][4]$ are defined as the qubit layers.

\(^3\)Simply speaking, loops should alter as little data as possible. Since every thread gets a copy of the runtime data and operates on its own variables, it can be very hard to merge them again.
2. Measuring syndromes has to be done very carefully here. We have to be strictly consistent with the qubit numbering. The numbering is illustrated in figure 3.3.1.

3. Calculating the distance $D$ between two qubits $Q_{xy}$, $Q'_{xy}$ on a toric square-octagon lattice can not be done with a simple Manhattan metric, since there are holes in it (see figure 3.3.2).
Chapter 3. Numerics

Regarding the holes, we obtain

\[
D = \min(|x_Q - x_{Q'}|, L - |x_Q - x_{Q'}|) \\
+ \min(|y_Q - y_{Q'}|, L - |y_Q - y_{Q'}|) \\
+ 2 \times (1 - \delta_0(|y_Q - y_{Q'}|)) \times (\mod(1 + x_Q, 2)) \\
+ 2 \times (1 - \delta_0(|x_Q - x_{Q'}|)) \times (\mod(1 + y_Q, 2))
\] (3.3.2)

for the distance on the lattice with

\[
\delta_0(x) = \begin{cases} 
1, & x = 0 \\
1, & \text{else}
\end{cases}
\] (3.3.6)

being the \(\delta_0\)-function. Simply speaking, if two qubits \(Q_{xy}, Q'_{xy}\) have the same odd \(y\) coordinates, the distance in \(x\) increases by two\(^4\). This works analogously for the distance in \(y\). To get an idea what this metric feels like, we can take a look at figure 3.3.3, which represents a circle on the lattice. The space is not isotropic anymore.

Figure 3.3.3 – A circle in the Manhattan metric with holes - Points with the same distance from the center are marked Red.

4. Error correction is not as easy as before, because we need an algorithm to walk on the lattice from one defect to another. Regarding certain properties of the path, different sets of qubits have to be corrected. Here, the “Long Walk To Freedom”-algorithm is proudly presented:

\(^4\)If we chose \([0][0][0]\) to be a plaquette vertex, both coordinates would have to be even to increase the distance by two.
A Long Walk to Freedom Algorithm: Let us choose to
- first correct errors in $y$ direction (horizontal),
- correct the surrounding qubits from the octagons that lay in between and
- to always walk in positive $y$ direction\(^5\).

We can then identify 4 different qualities of a single step $s$ in the path that have to be considered. The beginning coordinate $A$ can either be a square □ or an octagon ○, as well as the end coordinate $B$:

$$A, B \in \{□, ○\}. \tag{3.3.7}$$

The future $x$ (vertical) direction $\lambda$ can either be up $\uparrow$, down $\downarrow$, or neither, in which case it is also considered up by default:

$$\lambda \in \{\uparrow, \downarrow\}. \tag{3.3.8}$$

Last, but not least we have to know at which part of the path $\chi$ we are. There is the beginning $\triangleright$, a middle $\sim$ and the end step $\triangleleft$ to distinguish:

$$\chi \in \{\triangleright, \sim, \triangleleft\}. \tag{3.3.9}$$

Next, we must check which set of qubits ⊙ needs to be corrected. There are 6 different sets (see table 3.3.1) of qubits to identify, so

$$\odot \in \{\widehat{\odot}, x\odot, \odot x, \odot\odot, x\odot, \odot x\}. \tag{3.3.10}$$

The last step that needs to be done is to develop an algorithm deciding which ⊙ to choose regarding the properties of $s$. The algorithm is as follows:

$$\odot = \begin{cases} 
\begin{aligned} 
\text{for } x\odot: & \quad B = \odot \quad \chi = \triangleleft \quad \lambda = \uparrow \\
\text{for } x\odot: & \quad B = \odot \quad \chi = \triangleright \quad \lambda = \downarrow \\
\text{for } \odot x: & \quad A = \odot \quad \chi = \sim \quad \lambda = \uparrow \\
\text{for } \odot x: & \quad A = \odot \quad \chi = \triangleright \quad \lambda = \downarrow \\
\text{for } \odot: & \quad \chi = \sim \quad \lambda = \uparrow \\
\text{for } \odot: & \quad \chi = \triangleright \quad \lambda = \downarrow 
\end{aligned}
\end{cases} \tag{3.3.11}$$

So the correction algorithm walks the path from $y_0$ to $y_{q'}$, checks the properties of $s$ for each step that is made on an octagon, and correct the qubits defined by (3.3.11) and (3.3.10). For the vertical path in positive or negative $x$ direction, we simply tilt the lattice accordingly. The algorithm is structured so that the horizontal and vertical direction work analogously and independently.

To test the algorithm, two random points $Q$ and $Q'$ on the lattice were generated. Then, the algorithm applied $X$’s, according to 3.3.11 on the path from $Q$ to $Q'$. Afterwards, every check operator was executed and the amount of defects on the lattice was

\(^5\)This means that we will later walk in positive or negative $x$ direction.
Table 3.3.1 - Qubit sets that have to be corrected. Red marks the direction of the path, green marks the qubit sets that are corrected.

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Set of Qubits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Up</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Before Down</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Next Up</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Next Down</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Four Up</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Four Down</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
counted. Obviously, there should always be exactly 2 defects on the lattice during this procedure. The described procedure was executed $10^4$ times, every time exactly 2 defects appeared on the lattice which makes it very unlikely that the algorithm is incorrect.

- The rest of the simulation works without modifications.

### 3.3.2 Noisy Error Correction

#### Toric Code

1. The lattice is produced as before.

2. We prepare an $L \times L \times (T + 1)$ array\(^6\) with $T = L$ to store the parity check history $H$. Then, we evolve the lattice $T$ times generating qubit errors with probability $p$, measuring check operators with error probability $q$ and saving the history in $H$.

3. In the first slice, every syndrome is considered a defect (since it changed from no error to error). In every other slice, it is considered a defect only if the syndrome changed from $T - 1$ to $T$.

4. We can calculate the distance $D$ between two qubits $Q_{xyt}$, $Q'_{xyt}$ with a 3-dimensional Manhattan metric, since the dimensions are independent. The only thing to consider (if $p \neq q$) is the weight of the edges. Therefore:

   $$D = w_s \left( \min(|x_Q - x_{Q'}|, L - |x_Q - x_{Q'}|) \right) + w_s \left( \min(|y_Q - y_{Q'}|, L - |y_Q - y_{Q'}|) \right) + w_t |t_Q - t_{Q'}| \quad (3.3.12)$$

   with $w_s$ and $w_t$ being (3.2.10) and (3.2.11).

5. The minimum weight matching is can be done as usual. But we only correct paths which have more dimensions than time and whose defects do not both lie in the last slice, since it is generated error free.

6. We then do one round of error correction with perfectly executed syndrome measurements to get rid of the undetected errors.

#### Concatenated Toric Code

1. Obviously, the evolution of the lattice has to be altered considering different error probabilities. Therefore, qubit errors are generated with $p$, square syndrome check errors with $q_{□}$ and octagon syndrome check errors with $q_{□}$.

---

\(^6\)There are no errors generated in the last slice $T + 1$. 
Chapter 3. Numerics

2. Unfortunately, we are unable to calculate the distance $D$ between two points in spacetime $Q_{xyt}$ and $Q'_{xyt}$ for every $p, q_\Box$ and $q_\bigcirc$. Since

\[ w_\Box = \log \left( \frac{1 - p}{p} \right), \]

\[ w_{t\Box} = \log \left( \frac{1 - q_\Box}{q_\Box} \right) \quad \text{and} \]

\[ w_{t\bigcirc} = \log \left( \frac{1 - q_\bigcirc}{q_\bigcirc} \right), \]

the spacetime is partially anisotropic and not blessed with a metric anymore, which can be seen in figure 3.3.4.

What we can do is implementing a Dijkstra-algorithm\(^7\) to calculate the distance between those points in spacetime. We construct a graph adding edges properly weighted with (3.3.13), (3.3.14) and (3.3.15). There are two different ways of doing that: Either calculate the distances new for every iteration and every vertex, or calculate every distance from every point to every other before the $N$ iterations and save it in an array so that it has not to be calculated again. We are now confronted with a trade off dilemma between CPU time and RAM that is needed

\(^7\)A Dijkstra algorithm walks through the whole graph and saves every distance to every vertex. This algorithm is very CPU-intensive, but simple. Regarding time constraints of this thesis, it was the easiest (although maybe slowest) choice to implement. Recommended is the $A^*$ algorithm (see [7]) that might work better. An $A^*$algorithm uses a knowledge-plus-heuristic cost function to determine the order in which the search visits vertices and only determines the distance between one start and one goal vertex.
to store the data. In a lattice, there are

\[ n = 2L \times 2L \times T = 4L^3 \]  \hspace{1cm} (3.3.16)

points in spacetime, where defects can occur. But since only errors create defects, about

\[ n_d = n \left( p + \frac{q_{\square} + q_{\square}}{2} \right) \]  \hspace{1cm} (3.3.17)

defects really do occur. Since the Dijkstra algorithm has to visit every vertex in the graph we can calculate how many vertices \( n_v \) there are to visit when doing \( N \) iterations: If calculated new every iteration, there are

\[ n_{v,\text{new}}(L) = n_d^2 N = 16L^6 \left( p + \frac{q_{\square} + q_{\square}}{2} \right)^2 N \]  \hspace{1cm} (3.3.18)

vertices that have to be visited. If we calculated all the distances before, there are only

\[ n_{v,\text{before}}(L) = n^2 = 16L^6 \]  \hspace{1cm} (3.3.19)

vertices to visit. With a defect probability of

\[ \left( p + \frac{q_{\square} + q_{\square}}{2} \right) = 0.06 \]  \hspace{1cm} (3.3.20)

and

\[ N = 10^5 \]  \hspace{1cm} (3.3.21)

the first implementation would be a CPU nightmare, as seen in figure 3.3.5.
Chapter 3. Numerics

Figure 3.3.5 – Vertices that need to be visited. The coefficients are \( p + \frac{q_\circ + q_\Box}{2} \) = 0.06 and \( N = 10^5 \).

But considering the RAM usage \( R \) during the iteration we need (with float precision (4byte) and using the symmetry of the matrix)

\[
R_{\text{new}}(L) = 16L^6 \left( p + \frac{q_\circ + q_\Box}{2} \right)^2 \frac{1}{2} \times 4 \text{ Byte},
\]

(3.3.22)

while the second implementation needs

\[
R_{\text{before}}(L) = 16L^6 \frac{1}{2} \times 4 \text{ Byte}
\]

(3.3.23)

of RAM. Since our biggest lattice should have \( L = 32 \), we will need roughly 35GB of RAM\(^8\) as seen in figure 3.3.6 only for storing all the distances! Since the university’s computing cluster\(^9\) has enough RAM we choose the second implementation and hope for the best.

\(^8\)At least since now we should consider learning how to use the computing cluster of the RWTH Aachen University.

\(^9\)cluster.rz.rwth-aachen.de

34
Chapter 3. Numerics

3.4 The Accuracy Threshold

What we obtain from the simulation are failure rates $f_L(p)$ versus error probabilities for different lattice sizes. The accuracy threshold $p_c$ is the $p$, at which every lattice has the same failure rate:

$$ p = p_c \iff f_L(p) = f_{L'}(p) : \forall L, L' (3.4.1) $$

We therefore fit second order polynomial curves through our data points to obtain $f_L(p)$ for all the lattices we simulate\(^\text{10}\). Then, every intersection $p_{cL,L'}$ is calculated. The final $p_c$ is an average of all the $p_{cL,L'}$’s.

3.5 The Efficiency

An interesting characteristic of an error recovery code is the efficiency regarding qubit overhead. We plot the number of failure rates $f_p(N_{\text{Qubits}})$ versus the number of qubits in the lattice. This is done for a static

$$ p = \frac{1}{10} p_c. \quad (3.5.1) $$

Such a plot shows, how efficient an error recovery code works with the available number of qubits.

\(^{10}\) Actually, the fitting function is $P_{\text{fail}}(x) = A + Bx + Cx^2$ with $x = (p - p_c)L\frac{1}{m}$, according to [17]. However, since we do not have enough data points we fit $P_{\text{fail}}(p) = A + Bp + Cp^2$ which works fine.
4 Results

Due to CPU and cluster time limitation, the number of iterations and the lattice sizes are not sufficient for a well determined $p_c$ for some plots. However, the results show a rough threshold estimation in several regimes.

4.1 Noise Free Error Correction

Toric Code  As seen in figure 4.1.1, the accuracy threshold for the noise free toric code is

$$p_{c, \text{perfect}}^{\text{toric}} \approx 10.3\%.$$  (4.1.1)

Figure 4.1.1 – Noise free accuracy threshold for the toric code. The data points are generated with $10^6$ iterations.
**Concatenated Toric Code**  As seen in figure 4.1.2, we obtain an accuracy threshold for the noise free concatenated toric code of

\[
p_c^{\text{concatenated, perfect}} \approx 10.1\%.
\]  

(4.1.2)

**Figure 4.1.2** – Noise free accuracy threshold for the concatenated toric code. The data points are generated with \(10^6\) iterations.

\[
\begin{align*}
\text{Lattice Size} & = 8 \\
\text{Fit} & \\
\text{Lattice Size} & = 16 \\
\text{Fit} & \\
\text{Lattice Size} & = 20 \\
\text{Fit} & \\
N & = 1000000 \text{ iterations} \\
P_{\text{threshold}} & = 10.1144
\end{align*}
\]
Chapter 4. Results

Noise Free Efficiency  The noise free efficiency plot shows the failure rate versus the number of qubits on the lattice for both the toric and the concatenated code.

![Toric VS. Concatenated Efficiency](image)

**Figure 4.1.3** – Efficiency for the noise free simulation. The data points are generated with $10^4$ iterations.
4.2 Noisy Error Correction

Toric Code  As seen in figure 4.2.1, the accuracy threshold for the noisy toric code is
\[ p_{c, \text{noisy}}^{\text{toric}} \approx 2.8\%. \] (4.2.1)

![Toric Code Threshold](image)

**Figure 4.2.1** – Noisy accuracy threshold for the toric code. The data points are generated with $10^6$ iterations.
Chapter 4. Results

Concatenated Toric Code  As seen in figure 4.2.2, we obtain an estimated accuracy threshold for the noisy concatenated toric code in the

\[ q_\square = p, \]  

(4.2.2)

\[ q_\# = p \]  

(4.2.3)

regime of

\[ p_{\text{c}}, q_\square = p, q_\# = p \approx 2.8\%. \]  

(4.2.4)

Figure 4.2.2 – Noisy accuracy threshold for the concatenated toric code. Here, the error probabilities are \( q_\square = p \) and \( q_\# = p \). The data points are generated with \( 10^4 \) iterations.
As seen in figure 4.2.3, the accuracy threshold for the concatenated toric code in the

\[ q_\triangle = p, \]  

(4.2.5)

regime is roughly

\[ p_{c, q_\square} = p, q_\square = 1.5p \approx 2.7\%. \]  

(4.2.7)

\[ q_\square = 1.5p \]  

(4.2.6)

\[ p_{c, q_\square} = p, q_\square = 1.5p \approx 2.7\%. \]  

(4.2.7)

Figure 4.2.3 – Noisy accuracy threshold for the concatenated toric code. Here, the error probabilities are \( q_\triangle = p \) and \( q_\square = 1.5p \). The data points are generated with 10⁴ iterations.
Chapter 4. Results

As seen in figure 4.2.4, the accuracy threshold for the concatenated toric code in the

\[ q_{\square} = p, \]  

(4.2.8)

\[ q_{\square} = 2p \]  

(4.2.9)

regime is

\[ p_{\text{concatenated, noisy}}^{c, q_{\square} = p, q_{\square} = 2p} \approx 2.3\%. \]  

(4.2.10)

Figure 4.2.4 – Noisy accuracy threshold for the concatenated toric code. Here, the error probabilities are \( q_{\square} = p \) and \( q_{\square} = 2p \). The data points are generated with \( 10^4 \) iterations.
Noisy Efficiency  The noisy efficiency plot shows the failure rate versus the number of qubits on the lattice for the toric code and for various regimes of the concatenated toric code.

Figure 4.2.5 – Efficiency for the noisy simulation. The data points are generated with $10^5$ iterations.
5 Conclusion

5.1 Results

Although all thresholds are only rough estimates due to time and CPU constraints, the results give a first glimpse of an idea how well the \([4, 2, 2]\)-concatenated code performs and which regimes might be of further interest.

5.1.1 Noise Free Error Correction

For the noise free simulation, we obtain

\[
\begin{align*}
 p_{\text{toric, perfect}} &\approx 10.3\% \\
 p_{\text{concatenated, perfect}} &\approx 10.1\%.
\end{align*}
\]

The result from [17],

\[
p_{\text{toric, perfect}} = 10.31
\]

could be reproduced. The result for the \([4, 2, 2]\)-concatenated toric code is only slightly worse than the threshold of the toric code. Regarding qubit overhead, the concatenated code is about as efficient as the toric one in the noise free regime.

5.1.2 Noisy Error Correction

For the noisy simulation, we obtain

\[
\begin{align*}
 p_{\text{toric, noisy}} &\approx 2.8\% \\
 p_{\text{concatenated, noisy}} &\approx 2.8\% \\
 p_{\text{concatenated, noisy}} &\approx 2.7\% \\
 p_{\text{concatenated, noisy}} &\approx 2.3\%.
\end{align*}
\]

The result from [17],

\[
p_{\text{toric, noisy}} = 2.93\%
\]

could be reproduced very closely. The result for the \([4, 2, 2]\)-concatenated toric code in the (5.1.5) regime is as good as the toric threshold. Although it is unlikely that we are able to perform 8-qubit parity checks with the same fidelity as 4-qubit parity checks, this is an interesting result. The other regimes perform only slightly worse than the threshold of the toric code. This means information gain about good and bad syndrome
measurements cannot overtake information loss generated by more faulty measurements. Regarding qubit overhead in the noisy regime, the concatenated code is not as efficient as the toric code for small lattices. However, already 32 qubits in the \([4, 2, 2]\)-concatenated code are as efficient as 32 qubits in the toric code. The different values for \(q\) do not seem to matter a lot.

5.2 Outlook

5.2.1 The Simulation

Since the computation for the noisy regime is very CPU-intensive, there are different optimizations to recommend regarding the simulation:

- The current program can be altered very easily to generate and correct \(X\), \(Z\), and \(Y\) errors. This is not worth the effort. The lattice array should not consist of complex data types. Since we only generate one type of errors, we could work with a boolean lattice array which would speed up computation a lot.

- The Dijkstra algorithm is simply too costly, since it has to visit every node in the graph. Recommended is the \(A^*\) algorithm which works with knowledge-plus-heuristic cost function and only determines the distance between two vertices of interest.

- In the noisy, concatenated scenario, certain regimes with an easy relation between \(q\) and \(q\) could be chosen. Then, the distance between two vertices would be fast to determine.

- Since the error for a Monte Carlo data point is \(\sim \frac{1}{\sqrt{N}}\), at least \(N = 10^5 - 10^6\) iterations are needed for a useful determination of \(p_c\). Small lattices need even more iterations.

However, the programming code used in this thesis can be a starting point for further investigation of concatenated toric codes or similar architectures.

5.2.2 The \([4, 2, 2]\)-Concatenated Toric Code

Supplemental simulations of other \(q\) and \(q\) regimes can be performed. Also, analytical work could be done to predict the threshold of the recovery code.

All in all, the \([4, 2, 2]\)-concatenated toric code works very fine and might be a promising architecture to implement experimentally. The estimated accuracy threshold values evidence that the \([4, 2, 2]\)-concatenated toric code can serve as a scalable circuit-QED architecture.
List of Figures

2.1.1 The toric lattice ................................................. 9
2.1.2 Operators and their edges ....................................... 10
2.1.3 Trivial and nontrivial loops .................................... 11
2.1.4 The logical operators: Blue indicates \( X \) errors, Red indicates \( Z \) errors. 12
2.2.1 Defects ......................................................... 13
2.2.2 Successful decoding ............................................ 14
2.2.3 Failed decoding .................................................. 14
2.3.1 3D spacetime, taken from [4]. ................................. 15
2.3.2 1D cross section through spacetime, taken from [4]. ....... 15
2.4.1 An overview of the failure rate for noise free concatenated toric code. The data points are generated with \( 10^3 \) iterations. 16
2.5.1 The Transmon: b) Optical micrography and a) schematic circuit diagram. The use of two Josephson junctions allows tuning of the effective Josephson energy via the external magnetic flux penetrating the superconducting loop. Graphic taken from [8]. 17
2.5.2 Eigenenergies \( E_m \) (first three levels, \( m = 0, 1, 2 \)) of the CPB Hamiltonian (1) as a function of the effective offset charge \( n_g \) for different ratios \( \frac{E_1}{E_C} \). All energies are given in units of the transition energy \( E_{01} \) (evaluated at the degeneracy point \( n_g = 1 / 2 \)). The zero point of energy is chosen as the bottom of the \( m = 0 \) level. The sequence of plots highlights the exponentially increasing flatness of energy levels and the slow loss of anharmonicity as \( \frac{E_1}{E_C} \) is increased. The third graphic from left is the Transmon regime. Graphic taken from [8]. 17
2.5.3 Schematic layout of the circuit QED architecture. Two superconducting Transmons are fabricated inside a superconducting co-planar waveguide resonator. Graphic taken from [1]. 18
2.5.4 Transmons coupled to 3D cavities, taken from[13]. ............ 18
2.6.1 Replacing qubits by 4 (or more) Transmon qubits in 3D cavities. 19
2.6.2 Square-octagon lattice. The qubits got replaced by 4 Transmons in 3D cavities. \( X \) and \( Z \) square parity checks take place inside the cavities where they measure the 4-qubit’s parities. \( X \) octagon parity checks take place at the former lattice sites. \( Z \) octagon parity checks take place at the former lattice plaquettes. Octagons check the parities of the 8 neighboring qubits. 20
3.2.1 Logical operators in the square-octagon lattice .................. 23
3.2.2 The failure rate for \( L = 8 \) and different propagation times \( T \). We see that there is an upper limit for the failure rate. More time steps generate more errors, but also generate more information about the lattice state. 

3.3.1 Qubit numbering

3.3.2 Manhattan metric with holes

3.3.3 A circle in the Manhattan metric with holes - Points with the same distance from the center are marked Red.

3.3.4 The 3D spacetime of the square-octagon lattice. Red circles indicate octagon parity checks, Blue squares indicate square parity checks. For some weighted edges the shortest distance from one vertex to another might not lead along the direct path.

3.3.5 Vertices that need to be visited. The coefficients are \( \left( p + \frac{q_{\square} + q_{\bigcirc}}{2} \right) = 0.06 \) and \( N = 10^5 \).

3.3.6 Amount of RAM usage to store the distances. The coefficient is \( \left( p + \frac{q_{\square} + q_{\bigcirc}}{2} \right) = 0.06 \).

4.1.1 Noise free accuracy threshold for the toric code. The data points are generated with \( 10^6 \) iterations.

4.1.2 Noise free accuracy threshold for the concatenated toric code. The data points are generated with \( 10^6 \) iterations.

4.1.3 Efficiency for the noise free simulation. The data points are generated with \( 10^4 \) iterations.

4.2.1 Noisy accuracy threshold for the toric code. The data points are generated with \( 10^5 \) iterations.

4.2.2 Noisy accuracy threshold for the concatenated toric code. Here, the error probabilities are \( q_{\square} = p \) and \( q_{\bigcirc} = p \). The data points are generated with \( 10^5 \) iterations.

4.2.3 Noisy accuracy threshold for the concatenated toric code. Here, the error probabilities are \( q_{\square} = p \) and \( q_{\bigcirc} = 1.5p \). The data points are generated with \( 10^5 \) iterations.

4.2.4 Noisy accuracy threshold for the concatenated toric code. Here, the error probabilities are \( q_{\square} = p \) and \( q_{\bigcirc} = 2p \). The data points are generated with \( 10^5 \) iterations.

4.2.5 Efficiency for the noisy simulation. The data points are generated with \( 10^5 \) iterations.
List of Tables

3.3.1 Qubit sets that have to be corrected. Red marks the direction of the path, green marks the qubit sets that are corrected. 30


Acknowledgments

I want to thank Prof. Dr. Barbara Terhal for making this thesis possible. Especially her knowledge, her patience and her time explaining this area of physics to me were beyond compare. Her supervision was straight forward, but gave me enough space to develop myself.

Next, I want to thank Prof. Dr. David P. DiVincenzo for being the second assessor of my thesis.

Regarding the programming, I want to thank Dr. Martin Suchara for providing the basic programming code that saved a whole lot of time during the thesis.

Also, I am thankful for the short, but highly effective introduction to parallel computing from Alvaro Frank.

Last, but not least I want to thank Laura Platte for her proofreading and for coping with me during some times of the project.
Selbstständigkeitserklärung

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.
Aachen, den 09. September 2013