

# Fault-tolerant Quantum Computing

Bhaskara Marthi      Mark Whitney

November 17, 2000

When carrying out a computation on a physical circuit, there is the possibility of error. For example, with quantum circuits, there may be bit/phase flips or inadvertant measurements. Fault-tolerant computing deals with trying to carry out computations reliably in the presence of such noise.

The idea is to store all data in error-correcting codes and then to modify the gates in the circuit to work directly on this encoded data. We will also need some additional gates to perform error-correction after each step. The goal is, given encoded qubits as input, to produce output that, with high probability, can be corrected to the right answer.

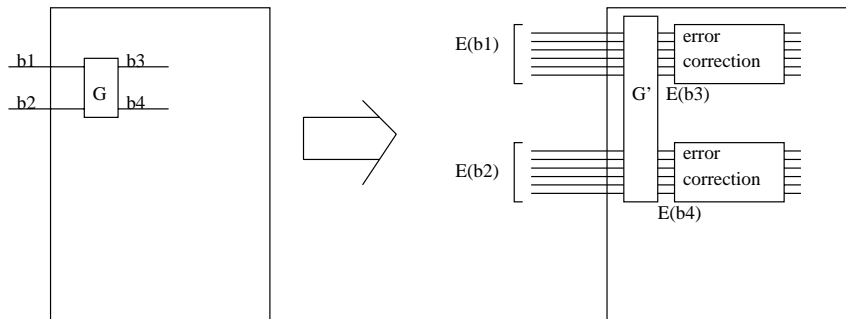


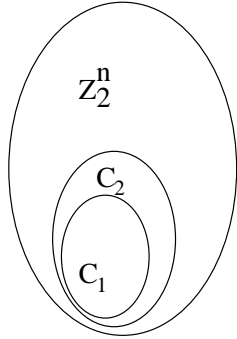
Figure 1: To do fault tolerant QC, we must convert the old quantum circuit to a circuit that will calculate over encoded data and then perform error correction.

We will make two assumptions. The first, which is not very restrictive, is that errors only occur at gates. The second, which is more restrictive, is that errors occur independently at each gate with probability  $\eta$ .

## 1 Shor's Result

This result, due to Shor, says that if  $\eta < \frac{1}{poly(log(S))}$  (where  $S$  is the circuit size), then we can perform fault-tolerant computation with high probability.

The first step is to figure out how to do error-correcting fault-tolerantly. Let  $C_1 \subset C_2 \subset Z_2^n$  be codes.



If  $C_1^\perp$  is an  $(n, n - k_1, d)$  code, and  $C_2$  is an  $(n, k_2, d)$  code, then we can encode  $k_2 - k_1$  bits, and correct  $d/2$  errors. Recall how this works : all errors are assumed to be combinations of bit and phase flips. The bit flips can be corrected classically using  $C_2$ . To correct phase errors, we first apply a Fourier Transform, which converts them into bit flips. We then correct these classically using  $C_1^\perp$ , and apply another Fourier Transform.

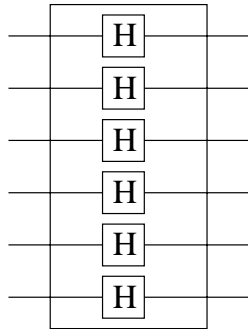


Figure 2: The FT over  $Z_2^n$ .

In doing the above, the Fourier Transform is fault-tolerant, because it works bitwise, so errors don't spread.

We just need to do the classical error corrections in a fault-tolerant way. If a codeword  $|c\rangle$  is transmitted as  $c' = |c + e\rangle$ , then the classical error-correction involves computing the syndrome  $(c + e)P$ , and reconstructing  $e$  from it. So we just need to compute this syndrome fault-tolerantly.

The  $i$ th bit of the syndrome is the sum mod 2, or the XOR, of  $c'_j$  where  $j$  ranges over all values such that  $P_{j,i} = 1$ . The obvious way to do this would be

to start with an ancilla bit  $a = |0\rangle$ , and then do successive CNOTs from each of the relevant bits  $c_j$  to  $|a\rangle$ .

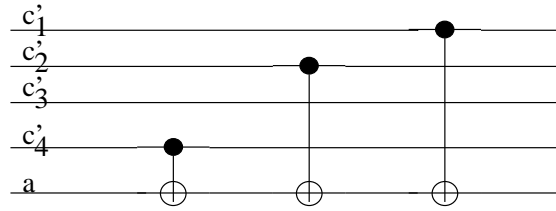


Figure 3: Computing the syndrome.

However, this circuit is not fault-tolerant, which can be seen by expressing it in the Hadamard basis.

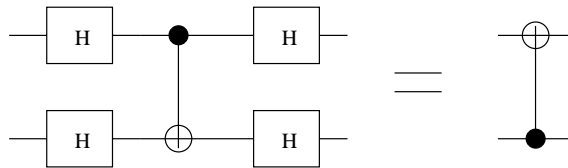


Figure 4: The CNOT in Hadamard basis is a flipped CNOT.

In this basis, as was shown in an earlier exercise, each of the CNOTs to  $|a\rangle$  becomes a CNOT *from*  $|a\rangle$ . So there is fanout - an error in  $a$  would propagate to many other bits.

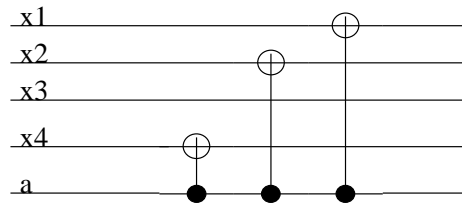


Figure 5: Highly undesirable fanout from the ancilla to codeword bits.

Here is a better way to compute the  $i^{th}$  bit of the syndrome. Let  $m$  be the number of 1's in the  $i^{th}$  column of  $P$ . Let  $|\phi\rangle$  be a uniform superposition over all  $m$ -bit strings having even Hamming weight, i.e an even number of 1's.  $|\phi\rangle$  can be prepared by applying  $H_2^m$  to the state  $|00\dots 0\rangle + |11\dots 1\rangle$ . Next,

apply a CNOT from each bit of  $c'$  to the corresponding bit of  $\phi$ . If the required bit is 0, then  $\phi$  will remain unchanged, while if it is 1, then  $\phi$  will now be a superposition over strings having odd Hamming weight. To determine which of these is the case, perform a measurement, and then check the parity of the Hamming weight of the result.

**Exercise :** Show that this procedure is fault-tolerant.

We now examine how small  $\eta$  needs to be for this to work. To do  $s$  steps of computation, we need to be able to correct  $\log(s)$  errors. Furthermore, we will require later on that the code satisfy  $\dim C_2 - \dim C_1 = 1$ . There exist codes that do this for which the codewords have length  $\log^2(s)$ . The number of CNOT gates added during the error-correction equal the number of 1's in the parity check matrix. Since this matrix has dimension  $O(\log^2(s))$  by  $O(\log^2(s) - \log(s))$ , an upper bound for the number of 1's is  $O(\log^4(s))$ . Finally, in order to make the probability of error smaller than some fixed  $\epsilon$ , we will need to repeat the computation  $O(\log s)$  times and take the majority vote. Multiplying, we find that  $\eta$  needs to be  $O(\frac{1}{\log^5(s)})$ .

Next, we need to modify the gates to work on error-corrected data. Of course, it suffices to do this for some universal set of gates. Assume that each qubit is encoded using codes  $C$  and  $C^\perp$ , where  $\dim C - \dim C^\perp = 1$ . So there are only 2 cosets of  $C^\perp$  in  $C$ , which is fine since we just want to encode one qubit.

Let  $|S_0\rangle = |C^\perp\rangle$  and  $|S_1\rangle = |C^\perp\rangle + |00\dots 1\rangle$  be the two codewords. Then, under the Fourier Transform

$$\begin{aligned} |S_0\rangle &\rightarrow \frac{1}{\sqrt{2}}(|S_0\rangle + |S_1\rangle) \\ |S_1\rangle &\rightarrow \frac{1}{\sqrt{2}}(|S_0\rangle - |S_1\rangle) \end{aligned}$$

Thus, we have implemented a Fourier Transform on an encoded qubit.

Next, consider the CNOT gate, which sends  $|S_a\rangle|S_b\rangle$  to  $|S_a\rangle|S_{a+b}\rangle$ . First note that

$$|S_a\rangle|S_b\rangle = \sum_{c \in C^\perp} |c+a\rangle \sum_{c' \in C^\perp} |c+b\rangle$$

If we just do a bitwise addition, this becomes

$$\begin{aligned} &\sum_{c \in C^\perp} |c+a\rangle \sum_{c' \in C^\perp} |c'+c+a+b\rangle \\ &= \sum_{c \in C^\perp} |c+a\rangle \sum_{c'' \in C^\perp} |c''+a+b\rangle \\ &= |S_a\rangle|S_{a+b}\rangle \end{aligned}$$

where we have used the fact that, since  $C^\perp$  is a subgroup,  $C^\perp = C^\perp + C^\perp$

**Exercise** Use a similar procedure to implement the phase-shift gate sending  $|S_a\rangle|S_b\rangle$  to  $(-1)^{a \cdot b}|S_a\rangle|S_b\rangle$ .

This set of gates is not yet universal. One way to make it universal is to add the Toffoli gate. However, implementing this fault-tolerantly is more tricky and we will not do it here.

## 2 Threshold calculation

The maximum probability of error to construct a fault tolerant quantum computer can be bounded to a constant,  $\eta < 10^{-6}$

To accomplish this we construct hierarchically fault tolerant compute and error correction stages, by replacing a elementary gate in a fault tolerant circuit with its fault tolerant equivalent construction.

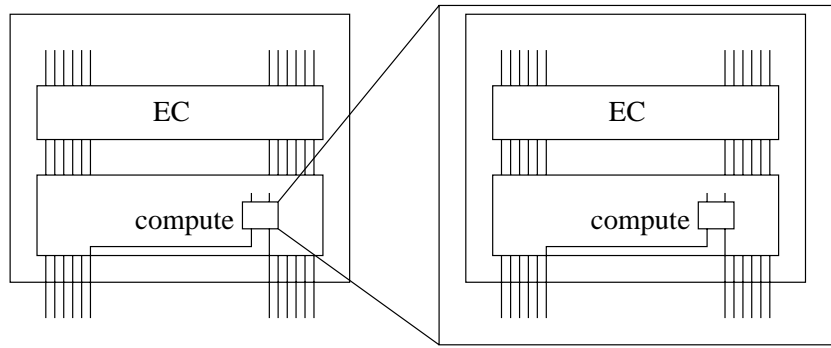


Figure 6: This circuit illustrates the hierarchical nesting of fault tolerant gates within individual gates of fault tolerant circuits.

In this hierarchical circuit, it may be desirable to have a more flexible body of codes to use. Go to polynomial codes and qubits ( $a \in \mathbb{F}_p$ ).

$$B : |a\rangle \rightarrow |a + 1 \pmod{p}\rangle$$

$$P : |a\rangle \rightarrow \omega^a |a\rangle$$

If  $F$  is the fourier transform over  $\mathbb{F}_p$ ,  $FPF^{-1} = B$  and so  $P$  and  $B$  span the space of all errors.

Use classical polynomial codes on the qubits to construct superpositions of codewords :

$$|S_a\rangle = \sum_{f: \deg f \leq d, f(0)=a} |f(\alpha_1), \dots, f(\alpha_m)\rangle$$

With this code we can correct  $\frac{m-(d+1)}{2}$  bit flip errors.

Next, go to the fourier domain and correct the phase errors by correcting the bit errors in the new domain. Since we know that the fourier transform of a polynomial code is another polynomial code this is not much of a problem.

$$|C_{a'}\rangle = \frac{1}{\sqrt{p}} \sum_b \omega^{ab} |S_{b'}\rangle$$

The following equality should also be made true:  $m - d - 1 = 2d$  so that  $S_a$  starts as  $\deg d$  and  $S_b$  is  $\deg 2d$ .

$$|a\rangle \rightarrow |a + c \pmod{p}\rangle$$

$$|S_a\rangle = \sum |f(\alpha_1) \cdots f(\alpha_m)\rangle$$

$$|a\rangle |b\rangle \rightarrow |a\rangle |a + b\rangle$$

$$|S_a\rangle |S_b\rangle \rightarrow \sum |f(\alpha_1) \cdots f(\alpha_m)\rangle + \sum |g(\alpha_1) \cdots g(\alpha_m)\rangle = |S_a\rangle |S_{a+b}\rangle$$

Here, pitwise addition of elements in  $S_a$  and  $S_b$  corresponds to the addition of the components in the 2 polynomial code words.

### 3 Degree Reduction

Observe the action of the Toffli gate:

$$|a\rangle \xrightarrow{F.T.} \frac{1}{\sqrt{p}} \sum_p \omega^{ab} |b\rangle$$

$$|a, b, c\rangle \rightarrow |a, b, c + ab \pmod{p}\rangle$$

If a the naive fault-tolerant construction of the gate is used, the  $ab$  term introduces a doubling of degrees in the polynomials of the associated codeword for the 3rd bit. In order to counteract this degree increase, a degree reduction operation must be integrated into the Toffli gate.

From the properties of the fourier transform, it is know that if the polynomials increase in degree in the base domain, their degrees decrease in the fourier domain.

The degree reduction is performed by mapping each polynomial  $f(\alpha_i)$  to a set of lower dimensional polynomials  $g_i(\alpha_1) \cdots g_i(\alpha_m)$ . These lower dimensional polynomials have the property that each  $g_i$  polynomial at zero maps to  $f(\alpha_i)$ .