

①

FAULT TOLERANCE

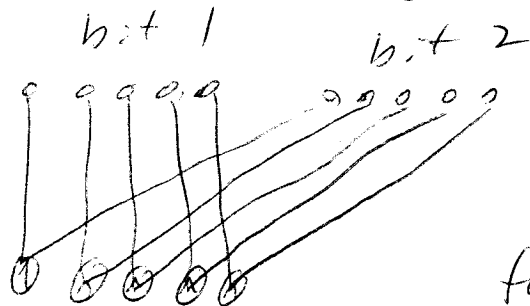
For classical computers, the question of fault tolerance came up in the 1950's, shortly after the first working computers were built. It was suggested that large computers would be inherently unreliable, because any sufficiently long sequence of logic gates would be sure to fail, as it is impossible to build perfectly reliable components.

Von Neumann refuted this argument by proving that it was possible, with a modest amount of overhead, to build reliable computers out of unreliable components. In fact, he caused considerable controversy by suggesting that the existence of such techniques was also proven by the existence of biological brains.

Von Neumann's technique was to replicate every bit in the computer several times, and to periodically compare these identical bits, and verify their states by setting them to the majority. While this technique will help to inspire the techniques for quantum computation, it is unable to be

(2)

adapted directly. How it fails is quite interesting.



von Neumann's technique for doing an AND (1) gate. You carry 5 copies of each bit around, and periodically compare them, settling any disagreements by using a majority vote.

Initializing k copies of each qubit is feasible, as is doing k parallel executions of the circuit. Where the technique breaks down is in the comparison of the of the different bits (this can be done to some extent, though, by switching a pair of computational states and checking whether they are identical), and, to a much greater degree, by the inability to replace a failed computation by a copy of a good one, as this would violate the no-cloning theorem.

③

How about using quantum error correcting codes for fault tolerance.

Clearly good for storing and transmitting data. To compute fault-tolerantly with codes, we need to do two extra things -

- a) We need to correct errors fault-tolerantly. That is, we need to be able to correct errors with noisy gates without causing more errors than we correct.
- b) We need to be able to compute fault-tolerantly with encoded qubits.

If we decode the qubits in order to compute, we expose them to error. This won't work. Need to compute on encoded qubits without decoding them.

(4)

Error model.

We will need some model for errors. Easy error model: assume errors only affect one qubit and one gates and are independent.

Is this realistic? Clearly not. Is it good enough to work in practice? Lots of controversy about this point. If we assume arbitrary errors, this error model doesn't work even on classical computers. (e.g. no defense against a bomb going off near your computer; it may scramble all your qubits at once).

On the other hand, if the classes of multi-qubit errors are readily classifiable, you may be able to devise fault-tolerant circuits that protect against them.

(5)

We can use CSS codes to provide fault tolerance.

Recall a CSS code was based on two classical codes
 $0 \subset C_2 \subseteq C_1 \subseteq \mathbb{F}_2^n$.

We will assume $C_2 = C_1^\perp$, and call C_1 by C .

Encode state $v \in C/C_2^\perp$ by

$$|v+C^\perp\rangle = \frac{1}{|C^\perp|^{1/2}} \sum_{w \in C^\perp} |v+w\rangle$$

We will assume

$\dim C = \dim C^\perp + 1$,
so only one qubit encoded.

All $v \in C^\perp$ have $\text{wt}(v) \equiv 0 \pmod{4}$
All $v \in C - C^\perp$ have $\text{wt}(v) \equiv 3 \pmod{4}$

6

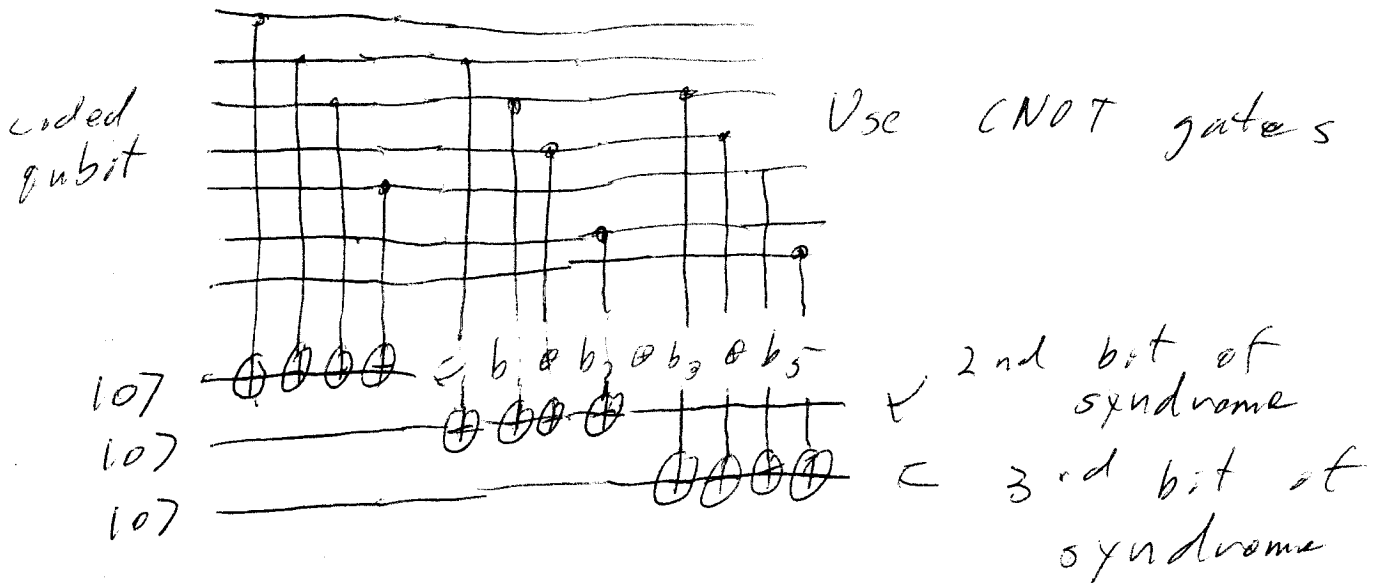
How do you correct errors in ~~Q~~ a code C . To correct bit (σ_x) errors, do this the same way you do for a classical code. Compute syndrome. ~~Do this~~

Hamming code

G generator matrix $\begin{cases} 1110100 \\ 0111010 \\ 0011101 \\ 1111111 \end{cases}$

$G^\perp = H = \begin{matrix} 1110100 & b_1 \oplus b_2 \oplus b_3 \oplus b_5 \\ 0111010 & b_2 \oplus b_3 \oplus b_4 \oplus b_6 \\ 0011101 & b_3 \oplus b_4 \oplus b_5 \oplus b_7 \end{matrix}$
 generated by

First try for correction circuit.



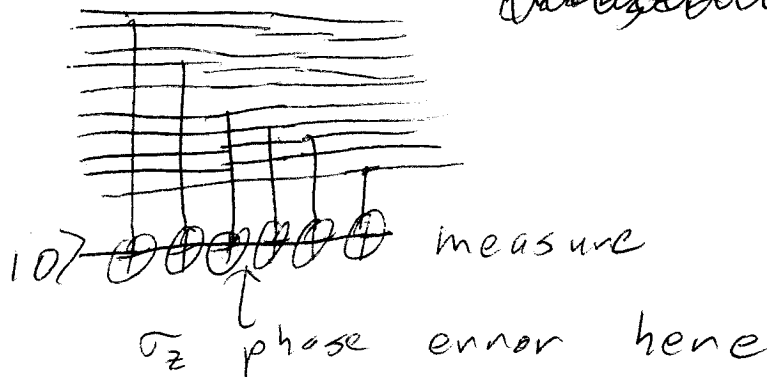
Measure bits of syndrome. These tell you what qubits to apply σ_x to so as to correct state.

7

Is this f.t.?


No.

Suppose we have six-bit ^{XOR in} syndrome
~~computation~~



Recall CNOT in σ_z basis
reverses target and control bit.

A σ_z error in middle of syndrome
computation applies σ_z to the
last three qubits in syndrome

(in phase basis, )

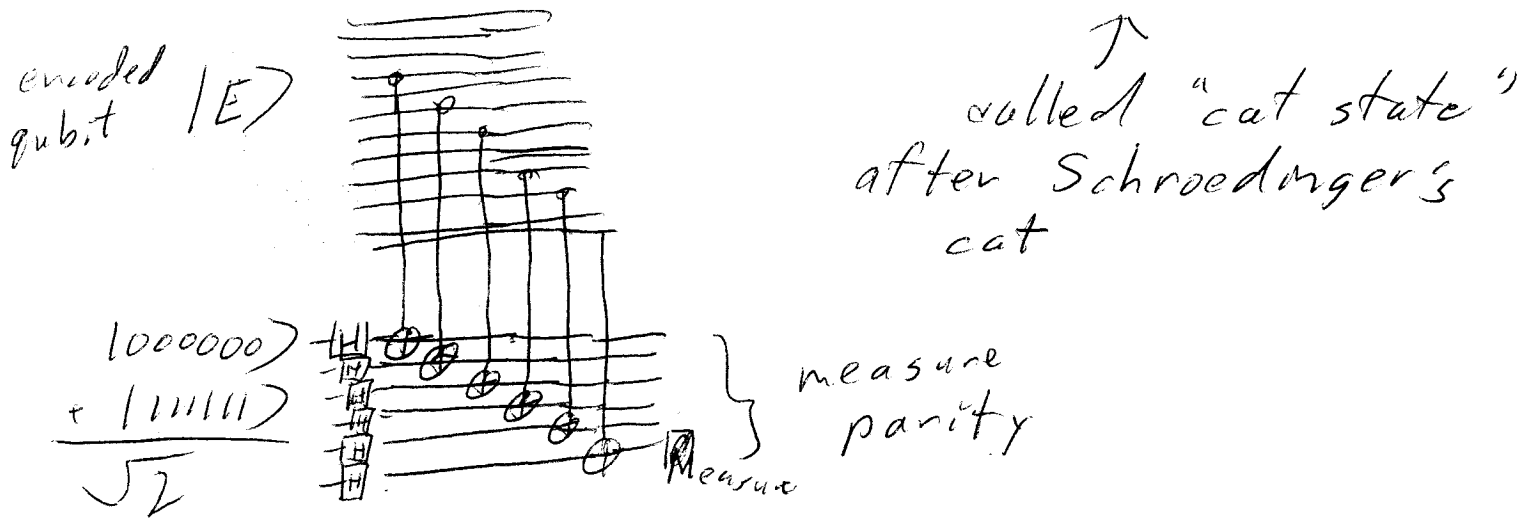
One gate error affects 3 qubits.
This is bad.

⊛ If we applied σ_z to all 6 qubits
in syndrome bit, this would not
affect encoded state.)

8

How do we fix this?

We want to find parity of bits in syndrome.
We use the state $\frac{1}{\sqrt{2}}(|000000\rangle + |111111\rangle)$



$$H^{\otimes 6} \left(\frac{|000000\rangle + |111111\rangle}{\sqrt{2}} \right) = \frac{1}{\sqrt{2}} \sum_{wt(s) \text{ even}} |s\rangle$$

So if everything works, we measure the parity of the syndrome bit.

What about back action on encoded state.

$|000000\rangle$ does nothing
 $|111111\rangle$ applies σ_z to each bit in syndrome calculation
 does nothing

9

Two questions left.

How do we prepare $\frac{1}{\sqrt{2}}(|000000\rangle + |111111\rangle)$?

What if we goof preparing it?

What we do is prepare it, and then verify we've prepared the right state by computing XOR of two qubits in this state.

This check ensures we don't have components in our state with \tilde{n} half zeros and \tilde{n} half one's. (or more than a few of both 0's and 1's).

These are the only components of the superposition that would give rise to back action that would harm our encoded state.

If we get the wrong value of the bit of the syndrome, that's o.k. We can compute enough parity check bits to obtain redundancy, and not correct our state until we're sure we know the error.

10

What about doing gates on encoded bits. It's clear we can't do arbitrary gates this way. But we don't need to.

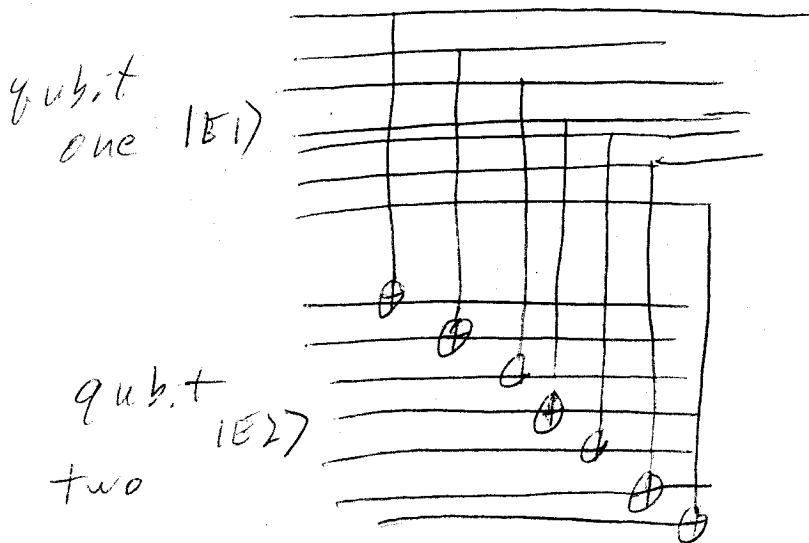
Only need H , $(\cdot i)$, Toffoli,

Recall our CSS code

$$|v+c\rangle = \frac{1}{|C^\perp|^{1/2}} \sum_{w \in C^\perp} |v+w\rangle$$

$v = \text{~~0~~ } 0^n$ or $v = 1^n$, since we only encode one qubit.

~~D~~ Do CNOT bitwise



This does CNOT on encoded qubits

11

$$|v + C^\perp\rangle = \frac{1}{|C^\perp|^{1/2}} \sum_{s \in C^\perp} |v+s\rangle$$

$$|w + C^\perp\rangle = \frac{1}{|C^\perp|^{1/2}} \sum_{t \in C^\perp} |w+t\rangle$$

$$\frac{1}{|C^\perp|} \sum_{s \in C^\perp} \sum_{t \in C^\perp} |v+s\rangle |w+t\rangle$$

Do bit wise CNOT

$$\frac{1}{|C^\perp|} \sum_{s \in C^\perp} \sum_{t \in C^\perp} |v+s\rangle |v+s+w+t\rangle$$

$$= \frac{1}{|C^\perp|} \sum_{s \in C^\perp} \sum_{t \in C^\perp} |v+s\rangle |v+w+t\rangle$$

$$= \frac{1}{|C^\perp|} \sum_{s \in C^\perp} \sum_{t \in C^\perp} |v+s\rangle |v+w+t\rangle$$

CNOT applied to $|w\rangle|w\rangle$ If $v = 0^n$, $v+w = w$ If $v = 1^n$, $v+w = \neg w$ (NOT w)

12

Similarly,

$$|E0\rangle = \frac{1}{\sqrt{C+1}} \sum_{w=C+1} |w\rangle \quad \leftarrow \text{all weight } 4$$

$$|E1\rangle = \frac{1}{\sqrt{C+1}} \sum_{w=C-C+1} |w\rangle \quad \leftarrow \text{all weight } 3 \pmod{4}$$

apply $(^i)$ bit wiseget $(^i - i)$ applied to encoded bitApply H bit wise

get H applied to encoded bit.
 Takes Fourier transform of $|E0\rangle$ to get $|E0\rangle + |E1\rangle$

This bit wise processing works for all operations in Clifford group.

Not good enough: need one operation outside Clifford group.

$$\left(\begin{matrix} ^i \\ \frac{1+i}{\sqrt{2}} \end{matrix} \right) \text{ or } \left(\begin{matrix} | \\ | \\ | \\ | \\ | \\ | \\ | \\ | \end{matrix} \right) \text{ Toffoli gates}$$

$\pi/8$ gate

(13)

Why is bitwise processing fault tolerant.

Any error in bit k cannot spread outside bit k of any encoded bit.

So a one-gate error will never disturb more than one qubit in each encoded state. Can be fixed by a code that corrects t errors if $t \geq 1$.

We also need to be able to initialize qubits to an encoded $|0\rangle$

$$|E0\rangle = \frac{1}{|C|^{1/2}} \sum_{w \in C} |w\rangle$$

We can do this in essentially the same way as we do error correction. Think of $|E0\rangle$ as a D -dimensional quantum code. Compute its syndrome, and correct. We can initialize qubits in $\frac{1}{\sqrt{2}}(|E0\rangle + |E1\rangle) = \frac{1}{|C|^{1/2}} \sum_{w \in C} |w\rangle$ in the

same way.

15

If it's $|E0\rangle$, the second qubit
is in the state
$$\alpha|E0\rangle + e^{i\pi/4}\beta|E1\rangle$$

If it's $|E1\rangle$, the second encoded qubit
is in the state
$$\alpha|E0\rangle - e^{i\pi/4}\beta|E1\rangle$$

and we can correct it by applying

$$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

to the encoded qubit, an operation
we can do fault-tolerantly b.t.wise.

Now, how do we create $\frac{1}{\sqrt{2}}(|E0\rangle + e^{i\pi/4}|E1\rangle)$?

We note that the eigenvalues of

$$\begin{pmatrix} 1 & -i \\ 1 & -i \end{pmatrix} \text{ are } \frac{1}{\sqrt{2}}(1+i), \text{ with e.vector } \begin{pmatrix} 1 \\ \frac{1+i}{\sqrt{2}} \end{pmatrix}$$

$$\text{and } -\frac{1}{\sqrt{2}}(1-i) \text{ with e.vector } \begin{pmatrix} 1 \\ -\frac{1+i}{\sqrt{2}} \end{pmatrix}$$

If we can create an eigenvector
of $\begin{pmatrix} 1 & -i \\ 1 & -i \end{pmatrix}$ acting on an

10

encoded qubit, we are done.

Idea: create state

$$\frac{1}{\sqrt{2}} (|0000000\rangle + |1111111\rangle) |E0\rangle$$

Use a controlled $\begin{pmatrix} 1 & i \\ & 1 \end{pmatrix}$ gate

on second qubit to get

$$\frac{1}{\sqrt{2}} (|0^k\rangle |E0\rangle + |1^k\rangle i |E1\rangle)$$

Apply $\frac{1+i}{\sqrt{2}}$ phase to $|1^k\rangle$
1 phase to $|0^k\rangle$

$$\text{get } \frac{1}{\sqrt{2}} (|0^k\rangle |E0\rangle + |1^k\rangle e^{i\pi/4} |E1\rangle)$$

measure in $\frac{1}{\sqrt{2}} (|0^k\rangle + |1^k\rangle)$

$\frac{1}{\sqrt{2}} (|0^k\rangle - |1^k\rangle)$ basis.

This can be done by measuring ~~number~~ ^{parity} of $|1\rangle$'s in complementary basis.

Why is this fault-tolerant?

Because the desired state is an eigenvector of $\begin{pmatrix} 1 & i \\ & 1 \end{pmatrix}$,

12

if we have a superposition of this eigenvector and its orthogonal encoded state, measuring the state with a basis that looks something like

$$\alpha |0^7\rangle + e^{i\theta} \beta |1^7\rangle$$

instead of $\frac{1}{\sqrt{2}}(|0^7\rangle + |1^7\rangle)$

will leave our desired state in superposition.

Most likely measurements will look something like that.

The same technique works for Toffoli gates: prepare state

$$\frac{1}{\sqrt{2}} (|E00\rangle|E00\rangle + |E0\rangle|E1\rangle + |E1\rangle|E0\rangle - |E1\rangle|E1\rangle)$$

$$\frac{1}{2} (|E000\rangle + |E010\rangle + |E100\rangle + |E111\rangle)$$

where $|Eabc\rangle$ is tensor product of encoded a , encoded b , encoded c .

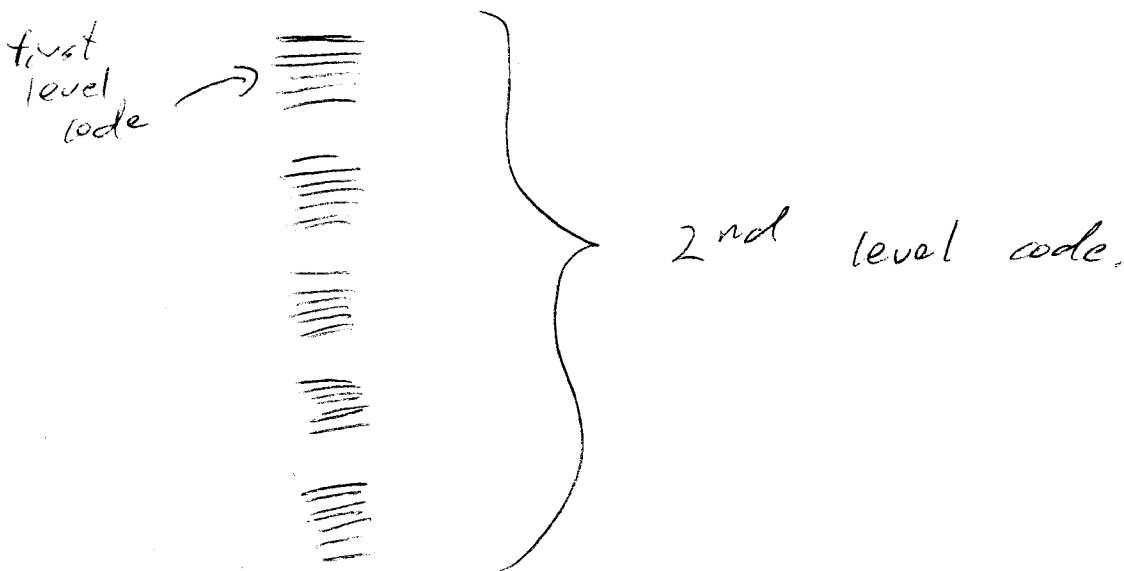
18

Finally, we need to show we can do n steps, n large with low error.

Problem: if we use a small number of qubits in our code, we have a relatively large chance of making an error.

If we use too many qubits in our ~~code~~ code, we need a low error rate to perform one encoded operation.

Solution: concatenate



19 For n steps, we need $\log \log n$ levels of concatenation. This gives $e^{c \log \log n} = (\log n)^c$ overhead.

In practice, two levels of concatenation probably suffice, and three are surely enough.

Error p at k^{th} level

\Rightarrow error Cp^2 at $k+1^{\text{st}}$ level

Takes $O(\log \log n)$ levels to get error probability $\leq \frac{1}{n}$.