Lattice Surgery

So far, we have described high-rate codes and also how universal quantum computing can be executed as a sequence of nondestructive Pauli-product measurements. Next we want to consider fault-tolerant implementation of Pauli-based computation using high-rate codes. But before we get to that, we'll discuss FTQC in the surface code.

In PBC, inputs are $|T\rangle$ states. For now, we'll suppose high quality $|T\rangle$ states are provided – in practice they'll be prepared by a method such as magic state distillation or cultivation. To compute reliably, then, it will suffice to perform FT nondestructive PPMs, chosen adaptively.

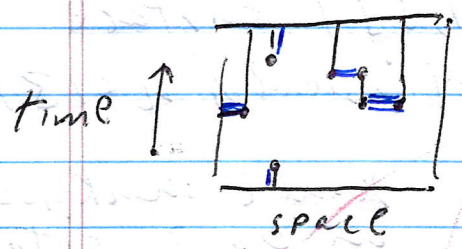
We'll consider two different methods, suited for different hardware modalities. The first is the surgery method, which is applicable to a platform in which qubits have fixed positions in a 2D layout with nearest neighbor connectivity (e.g. a superconducting processor). The second is the transversal method, which is applicable if Clifford operations such as CNOT and CZ are implemented transversally by moving encoded surface code blocks, e.g. in an atomic processor.

An advantage of the transversal method is that it substantially reduces the time-blowup due to FT — $O(1)$ rounds of syndrome extraction in between logical operations instead of $O(d)$ rounds for surgery (where d is code distance) — at the cost of somewhat increasing the physical qubit count, and increasing the complexity of (real-time) syndrome decoding, as well as requiring qubit mobility.

Syndrome decoding

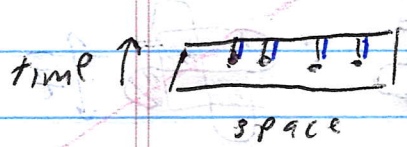
We discussed syndrome decoding in the surface code when there are both qubit errors and syndrome errors for the purpose of maintaining a quantum memory. If we want to compute, how many SE rounds should be performed in between logical gates (which can spread errors to other code blocks)? The standard view (to be revisited later) is that for a distance-d code we should do d SE rounds between operations.

Even for the case of quantum memory, we may use a "sliding-window" decoder that takes a finite number of SE rounds as input. Recall we may apply MWPM to the syndrome history in spacetime.



We match defects (endpoints of syndrome strings) to each other, or to spatial boundaries, or to past/future boundaries.

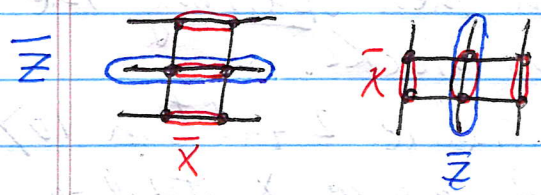
~~Match~~ Match to future boundary when we lack confidence in syndrome's authenticity. If the window is too short in time direction, we'll fail to correct too many authentic errors.



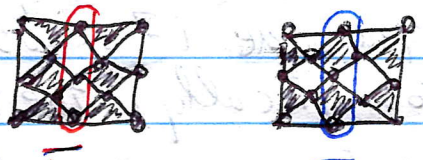
To enjoy the full benefit of distance d, window time extent should roughly match its spatial extent $\Rightarrow$ Old) SE rounds inside the window. We need to repeat this many times to gain sufficient confidence in the syndrome, before allowing block to contact other blocks.

Surface Code: Rotated and unrotated

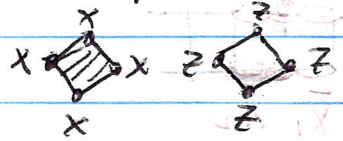
Consider $d=3$ "unrotated" $[[13, 1, 3]]$ surface code with qubits on edges



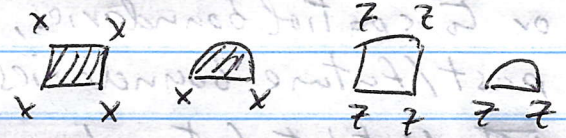
13 edges
6 plaquettes: 2 wt. 4 and 4 wt. 3.
6 sites: 2 wt 4 and 4 wt 3.



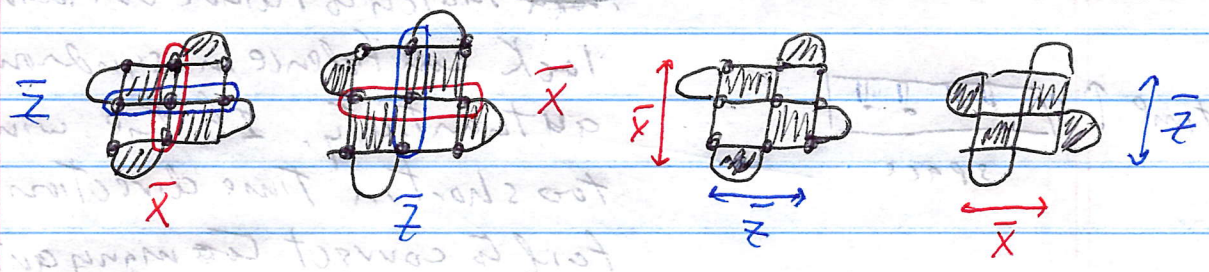
Alternative depiction with qubits on sites X and Z stab. on plaquettes.



If "rotated" $[[9,1,3]]$ surface code is more efficient ($n=9$ instead of $n=13$).



X and Z checkops collide on 0 or 2 sites.

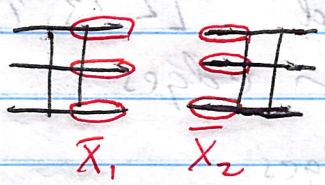


Now 4 X-type stab, 2 wt 4 and 2 wt 2

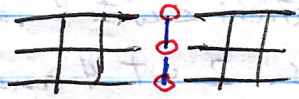
4 Z-type stab, 2 wt 4 and 2 wt 2

The boundary checkops determine how $\bar{X}$ and $\bar{Z}$ are oriented.

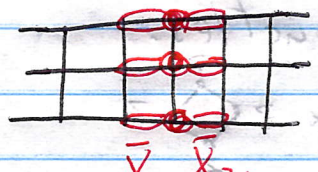
Suppose we want to measure XX or ZZ for two neighboring blocks using only geometrically local measurements. First, consider two blocks of unrotated surface code. Consider XX for two $d=3$ blocks (weight-6 operator)



we do the measurement by first merging and then splitting the two blocks.

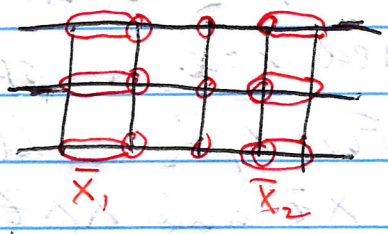


Initialize 2 qubits in $|0\rangle$ state, and measure X stabilizers of merged code (Z stabs are satisfied automatically). The three X_s outcomes are random. Their product is $X_1 X_2$!

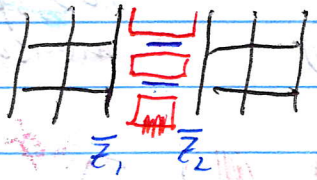


A measurement error in any of the site operators X_S will flip our value of $X_1 X_2$. We need to measure the syndrome $O(d)$ times to have confidence in the measurement outcomes. Finally, we split the blocks by measuring the two middle qubits in the Z basis, obtaining random outcomes $|0\rangle$ or $|1\rangle$. The $|1\rangle$ outcome flips the values of the boundary Z checkops in the blocks, which we can correct or absorb in to the Pauli frame.

Not that we can also obtain $X_1 X_2$ from the parity of a larger set of X_S outcomes which are bounded on left and right by X string logical operators:



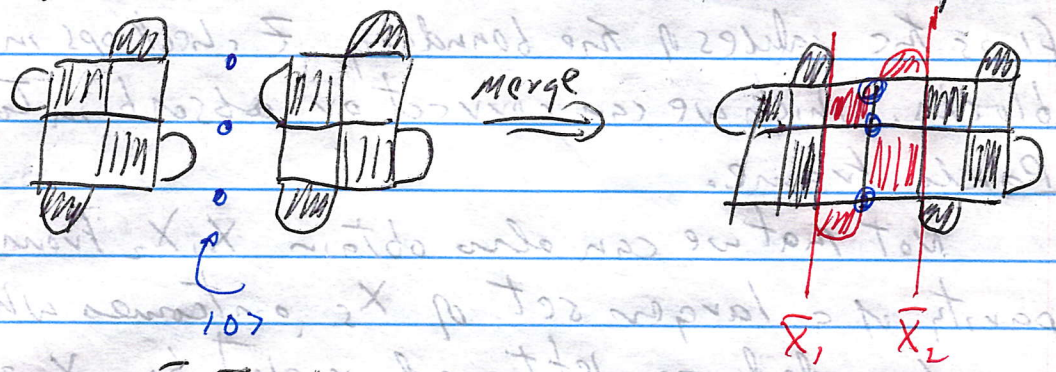
Measuring $Z_1 Z_2$ is the same procedure, but in the dual basis. Now we merge by introducing 2 qubits in $|+\rangle$ state and measure three Z_P operators. Their parity is $Z_1 Z_2$. Then split by measuring in X basis.



Not that e.g. when we measure $X_1 X_2$ by introducing

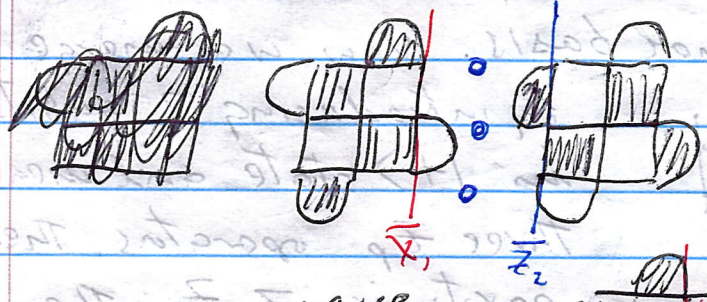
What if we want to measure mixed-type operators like $\bar{X}_1 \bar{Z}_2$? To do that we need to leave the CSS world — that is, measure stabilizers containing both X and Z. Let's consider this in rotated surface code.

First consider $\bar{X}_1 \bar{X}_2$ measurement. We initialize 3 qubits as $|0\rangle$ and measure 4 X-type stabilizers.

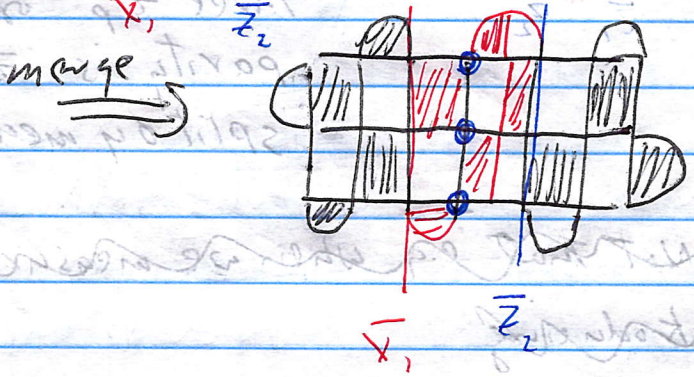


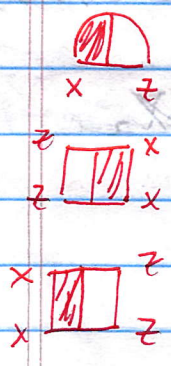
Now $\bar{X}_1 \bar{X}_2$ is the parity of the 4 checkops colored red. Can split by Z measurement.

For $\bar{X}_1 \bar{Z}_2$ we'll measure stabilizers like XZ and XXZZ.



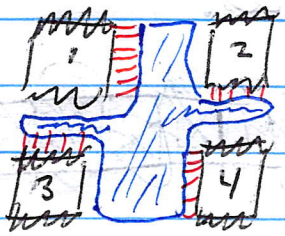
Can initialize as $|0\rangle$, split with Z measurement



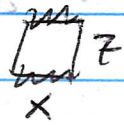


Again, a product of 4 checks, 2 of which are mixed, gives $\bar{X}_1 \bar{Z}_2$

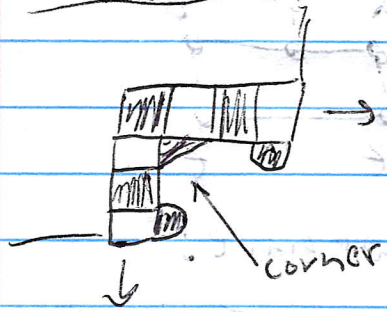
Using this trick of mixed stabilizers, we can measure high-weight ~~P~~ logical Paulis with a mix of Xs and Zs. We initialize ancillas which form a sea that can be connected to the X or Z boundaries of various blocks, so that desired logical operator is parity of check operators on terms, then split the blocks by measuring all the ancillas.



Measures: $Z_1 X_2 X_3 Z_4$

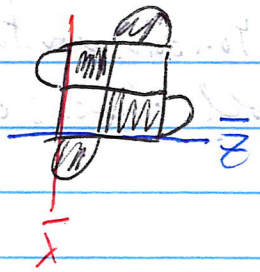


Note that a block can turn around at a corner where there is a weight-3 check.

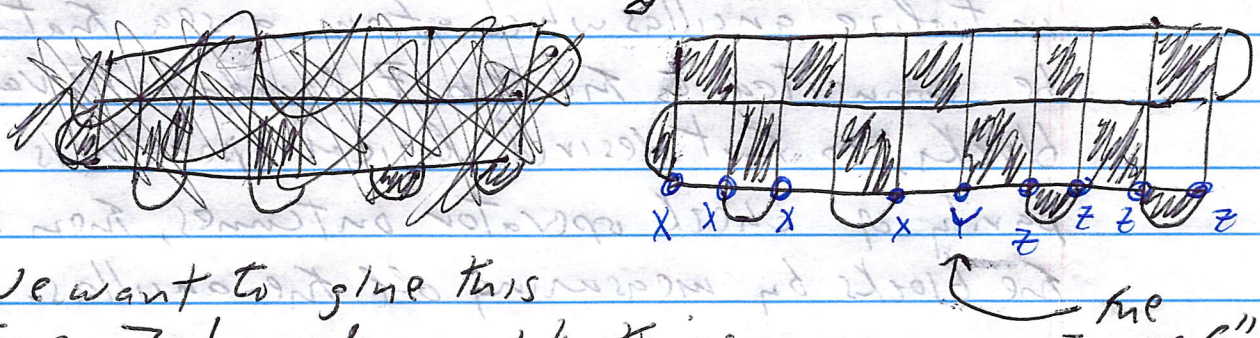


To measure a Pauli product that contains Y, though, we need a further trick: a "twist" defect.

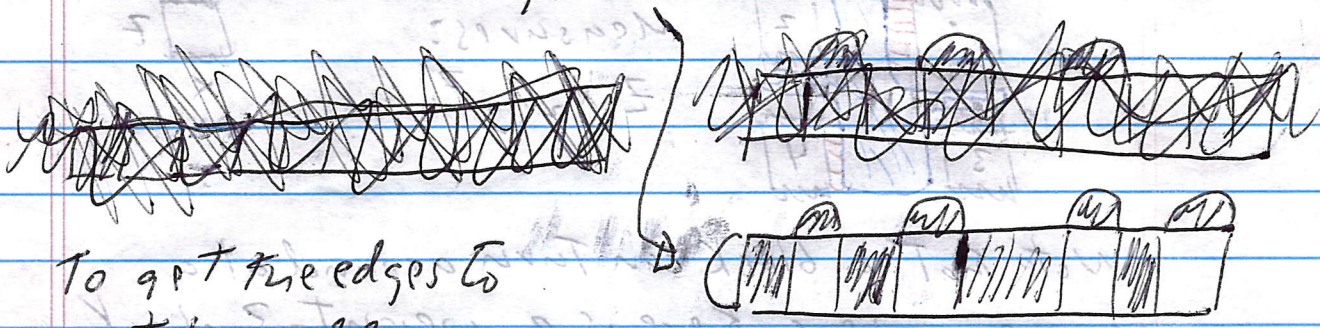
The logical $\bar{Y}$ is product of $\bar{X}$ and $\bar{Z}$ (up to a phase). On a $d \times d$ block, this is weight $2d-1$, with $d-1$ Xs, $d-1$ Zs, and a Y at the corner. To obtain this operator as a product of stabilizers, we need a stabilizer containing Y.



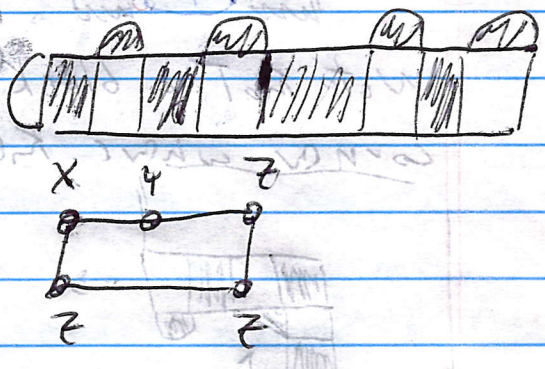
we can "straighten out the corner", with Z and X edges meeting where Y is inserted. It looks like this:



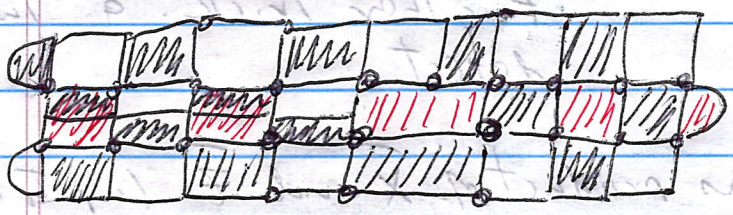
We want to glue this to a Z boundary, like this:



To get the edges to match we'll need a weight-5 stabilizer (the "twist").



When we merge the blocks we have:



(Here block means X, white means Z, and red indicates which checks are multiplied to yield the target logical operator.)

Take the product of checks colored red. This is $\bar{Y}_1 \bar{Z}_2$!

You can verify that all checks commute.