Lecture 16
27 May 2026
ph/cs 219

Measuring logical operators in high-rate codes

We have seen that universal quantum computation can be reduced to (1) Preparation of (high-quality) non-Clifford states, such as $|T\rangle$ states. (2) Measurement (fault tolerantly) of (possibly high-weight) logical Pauli product operators. This is called Pauli-based computation (PBC).

Taking magic-state preparation for granted, we discussed fault-tolerant Pauli product measurement in the surface code. We would like to generalize the lattice surgery method that works for the surface code to other qLDPC codes, including high-rate codes.

A crucial feature of the surface code is that syndrome measurement requires only geometrically local connectivity in a 2D layout, and lattice surgery is formulated to be compatible with that geometric constraint. In high-rate codes we already allow geometric nonlocality in syndrome measurement, so there is no reason to insist on local connectivity in logical processing. But other features of lattice surgery provide guidance.

Viewed broadly, lattice surgery involves these steps/features.

- Introduce additional ancilla qubits initialized in a product state.
- Define a "deformed code" whose checks act on both the original data qubits and the ancilla (including some checks that act on both).
- The deformed code is designed so that the target logical operator to be measured is a product of the (low-weight) check operators of the deformed code.
- The deformed code should be qLDPC (have low-weight checks) and should have a distance comparable to the distance of the original data code.
- We measure the checks of the deformed code multiple times to have high confidence in their value.
- We infer the value of the target logical by taking the parity of the outcomes of the appropriately chosen check operators.
- We measure the ancilla qubits to remove them from the deformed code and return to the original code.

(Before I spoke of a "merged code." Now I'm saying "deformed code" because in some cases we'll consider PPMs within a single code block rather than a product of Paulis in different blocks.)

There are two (related) ways of looking at lattice surgery, each suggesting a method for generalizing it, described in these references.

Williamson & Yoder, 2410.02213
"Low-overhead fault-tolerant quantum computation by gauging logical operators" (W-Y)

Ide, Gowda, Nadkarni, Dauphinais 2410.02753
"Fault-tolerant logical measurements via homological measurements" (Xanadu)

Gauging logical operators:

A quantum subsystem code is defined by a set of (typically low weight) generators which are not necessarily commuting, and the code stabilizer is the center of the gauge group (hence stabilizer generators are expressible as products of gauge generators). There are logical operators that commute with gauge group but are not contained in the gauge group ("bare" logical operators). We may also consider "dressed" logical operators, obtained by multiplying bare logical operators by elements of the gauge group. We don't try to protect the gauge qubits, rather the code protects the (bare) logical operators, which may have much

higher weight than the gauge generators. What is useful is that we can determine the (high-weight) stabilizers by measuring the (low-weight) gauge operators.

When we speak of "gauging logical operators" we mean that the deformed code may be regarded as a subsystem code, such that the high-weight target logical operator can be expressed as a product of low-weight gauge generators.

Example of subsystem code: Bacon-Shor code. Code is $[[9,1,3]]$.

$\bar{X} = \begin{pmatrix} X & X & X \\ I & I & I \\ I & I & I \end{pmatrix}$ $\bar{Z} = \begin{pmatrix} Z & I & I \\ Z & I & I \\ Z & I & I \end{pmatrix}$ are logical Paulis

Stabilizer:

$S_{X1} = \begin{pmatrix} X & X & X \\ X & X & X \\ I & I & I \end{pmatrix}$ $S_{X2} = \begin{pmatrix} I & I & I \\ X & X & X \\ X & X & X \end{pmatrix}$ $S_{Z1} = \begin{pmatrix} Z & Z & I \\ Z & Z & I \\ I & I & I \end{pmatrix}$ $S_{Z2} = \begin{pmatrix} I & I & I \\ I & Z & Z \\ I & Z & Z \end{pmatrix}$

Each wt. 6 stab generator is a product of 3 wt. 2 gauge generators, e.g.

$S_{X1} = \begin{pmatrix} X & I & I \\ X & I & I \\ I & I & I \end{pmatrix} \begin{pmatrix} I & X & I \\ I & X & I \\ I & I & I \end{pmatrix} \begin{pmatrix} I & I & X \\ I & I & X \\ I & I & I \end{pmatrix}$

Each gauge generator commutes with stabs and (bare) logicals $\Rightarrow$ we can determine stab generators by measuring low-weight gauge generators! "Fix the gauge" $\longrightarrow$ Shor code

Homological measurement:

Recall from Lecture 8 the connection between CSS codes and chain complexes

$$V_2 \underset{\delta_1}{\overset{\partial_2}{\rightleftarrows}} V_1 \underset{\delta_0}{\overset{\partial_1}{\rightleftarrows}} V_0$$

$$\partial_2 = H_Z^T, \quad \partial_1 = H_X, \quad \delta_1 = \partial_2^T = H_Z, \quad \delta_0 = \partial_1^T = H_X^T$$

Hence $\partial_1 \circ \partial_2 = H_X H_Z^T = 0$

$\delta_1 \circ \delta_0 = H_Z H_X^T = 0$

$$\mathcal{L}_Z = \frac{\ker(\partial_1)}{\mathrm{Im}(\partial_2)} = \frac{\ker(H_X)}{\mathrm{Im}(H_Z^T)}$$

commutes with X-stab and not in Z-stab.

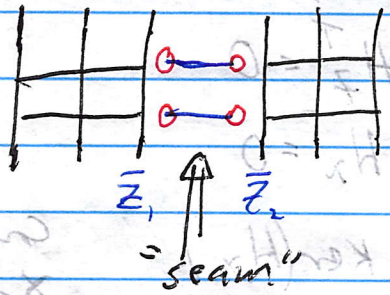
$$\mathcal{L}_X = \frac{\ker(\delta_1)}{\mathrm{Im}(\delta_0)} = \frac{\ker(H_Z)}{\mathrm{Im}(H_X^T)}$$

commutes with Z-stab and not in X-stab.

When we speak of "homological measurement" of a logical operator, we mean that the target logical (let's say a Z-Type operator) which is closed but not exact in the original code (in $\ker(\partial_1)$ but not in $\mathrm{Im}(\partial_2)$) becomes exact in the deformed code (in $\mathrm{Im}\,\partial_2'$, the image of deformed boundary operator). If the deformed code is qLDPC, this means the logical becomes expressible as a product of the

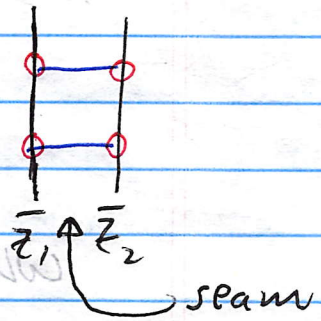
low-weight checks of the deformed code.

Consider lattice surgery in the surface code from the perspective of homological measurement, e.g. the measurement of $\bar{Z}_1 \bar{Z}_2$ on two blocks of the unrotated surface code.



we add the two ancilla qubits shown as horizontal blue edges, and sew the two blocks together at seam.

In the deformed code, 4 weight-3 site operators are promoted to weight-4 site operators, and 3 new plaquette operators are added. The target logical $\bar{Z}_1 \bar{Z}_2$ is the product of these 3 plaquette operators (the blue horizontal edges across the seam cancel out).



In this case, the logical really is the boundary of the set of plaquettes at the seam.

How to interpret this procedure as "ganging" the logical operator? We can say that we have added 3 gauge generators, the 3 new plaquettes, all of which fail to commute with checks of the original code, but their product yields $\bar{Z}_1 \bar{Z}_2$.

(27 May 2026 (7)

How to "gauge logical operators"

We follow G-T. Suppose without loss of generality that the target logical operator is of X-type. We'll introduce ancillas and deform the original code to a subsystem code which has X-type gauge generators, acting on the original qubits and the ancilla qubits, with low weight, as well as low-weight Z-type stabilizers acting only on ancillas. We want the target X-type logical to be a product of low-weight X-type gauge operators, and we want the distance of the deformed subsystem code (lowest weight of dressed logical operator) to be no less than distance of original code. It is also important that each qubit participates in O(1) gauge generators.

In the protocol for PPM we measure the gauge generators O(d) times. Because the code is a subsystem code, the X-type and Z-type generators do not commute and hence fluctuate during the d rounds. But the logical we decode is a ~~bare logical~~ gauge operator of the ~~subsystem~~ subsystem code (commutes with all gauge generators) and hence can be determined from the syndrome history (as in Bacon-Shor code).

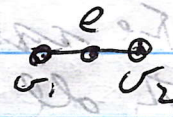
Measurement of X-Type logical operator L is achieved by a ~~subsystem~~ subsystem code associated with a graph $G(V,E)$ which is required to have certain specified properties. The vertices $v \in V$ of the graph correspond to qubits in the support of L. The edges in E have the property that all cycles in the graph are generated by a set $\underline{P}$ of cycles of constant length (here P stands for "plaquettes," not for Pauli – these short cycles are analogous to the plaquettes of the surface code). We associate an ancilla qubit with each edge $e \in E$.

To construct the deformed subsystem code, we add gauge generators associated with each $v \in V$ (each qubit in the support of L

$$A_v = X_v \bigotimes_{e \ni v} X_e$$

A product of X acting v and X acting on each of its neighboring edges. In physics terminology this may be called a "Gauss law operator." Note that

$$L = \bigotimes_v X_v = \bigotimes_v A_v$$



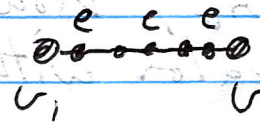
because each edge has two neighboring vertices, so two factors of X_e on edge e cancel out.

In addition there are gauge generators of Z-type, "flux operators" associated with all the short generating cycles on G

$$B_\ell = \bigotimes_{e \in \ell} Z_e$$

These are actually in the code stabilizer, they commute with all A_v because a cycle has support on an even number of edges neighboring v.

The Z-type ~~gauge~~ stabilizers of the original code may fail to commute with some of the A_v gauge generators. We know L commutes with the original stabilizer so the Z-type stab generator has support on an even ~~number~~ number of L's qubits — to be concrete suppose the number is 2.



We will assume that for any pair of vertices both in the support of any Z-type generator of the original code there is an edge path on the graph of constant length that connects them. Then we apply Z_e to each of these edges to obtain a new stabilizer that commutes with the gauge generators of the deformed code.

It is important that each qubit participates in O(1) gauge generators in that we want to be able to measure all the generators in constant depth.

For fault-tolerance, we would like the deformed code to have large distance. We introduced a lot of ancilla qubits and we worry that there could be logical operators in the deformed code that have lower ~~distance~~ weight than the distance of the original code.

We don't need to worry about Z logicals supported only on ancillas. To commute with all A_v, a Z operator needs to be a cycle of $G(V, E)$, and by construction all cycles are in the stabilizer, hence can't be nontrivial logicals.

More concerning are X operators with support on vertices. Can these be partially cleaned with the gauge generators $\{A_v\}$ resulting in a "dressed" logical operator with low weight?

Cleaning a product of X_v's using an A_v to remove each X_v, results in an operator with support on the edges in the boundary of the set where the original operator had support

$$\left(\bigotimes_{v \in S} X_v\right) \underset{\text{gauge equivalent}}{\sim} \left(\bigotimes_{e \in \partial S} X_e\right).$$

We'll suppose the graph $G(V,E)$ is an expander with the property:

$$|\partial S| \geq |S| \text{ for } |S| \leq |V|/2$$

In that case, the dressing cannot reduce the weight. A graph with this property is said to have Cheeger constant $h(G) \geq 1$.

To recap, we want our graph to have these properties:

- The graph has constant degree
- All cycles on G are generated by a sparse set of constant-length cycles
- Pairs of vertices that are in the support of the same Z-type stab generator of the original code are connected in G by a path of edges with constant length
- The Cheeger constant $h(G)$ is at least 1.

W-Y show that such a graph exists (we might need to add some additional "dummy vertices" beyond the support of L), and that it is not much larger than W, the weight of the target logical operator L. Specifically the sum of the number of vertices + the number of edges is

$$|E| + |V| = O(W \log W),$$

and so in particular the number of ancilla qubits used is $O(W \log W)$ - this is the qubit overhead of the measurement protocol. The time overhead is $O(d)$, the number of times we need to repeat the gauge generator measurements to achieve fault tolerance.

This W-Y protocol, based on gauging, is analyzed graph-theoretically. What will consider next, the Xanadu protocol, is analyzed homologically and based on a different idea! The chain map and its mapping cone.