Reducing the Spatial Overhead for Quantum Error Correcting Codes

Aarav Pabla
University of Maryland, College Park

Institute for Robust Quantum Simulation

Joint Center for Quantum Information and Computer Science

Abstract

Quantum computers theoretically promise large speedups compared to classical computers. However, just like classical computers, quantum computers are susceptible to errors. Why is it that damaged CDs can still be read just fine, if the scratch is not tool large? Optical drives have robust procedures such that a certain number of errors can be detected and fixed. Quantum error-correction aims to do the same for quantum computers. The hypergraph product code is a quantum error-correcting code that seems to offer close to the "best" parameters that characterize an error-correcting code. Despite this, under certain conditions, the code can use a lot of unnecessary qubits. We aim to reduce this qubit overhead while maintaining the code's useful parameters.

Background

A classical error correcting code is formed by checking the parity (even or odd) of certain bits. The parities checked can be modeled by a Tanner graph, where circular vertices are bits and rectangular vertices are checks. As shown in Fig. 1, if we know that the parity of bits 1 and 2 is even and 2 and 3 is odd, we can deduce that the bits are 001. Such checks can be described by a **parity-check matrix (PCM)**. In Fig. 1, the PCM is

$$H = \overbrace{\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}}^{\text{qubits}}.$$

A PCM defines an error-correcting code. A quantum error-correcting code uses parity checks as well but must check two for two types of errors: bit- (X) and phase-flip (Z) errors. The hypergraph product code (HGP) is made from taking the graph product of two classical Tanner graphs, where combining a parity-check and bit yields a check for X errors, combining a bit and parity-check yields a check for Z errors, and combining two bits or two parity checks yields a qubit. It turns out for HGPs where the classical codes are non-redundant, the qubits that come from combining two parity checks are not needed. Our work aims to get rid of these qubits.

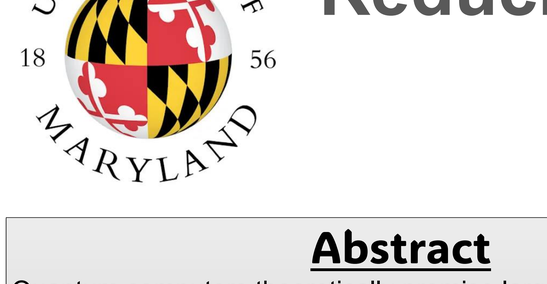

Figure 1: Tanner graph example.

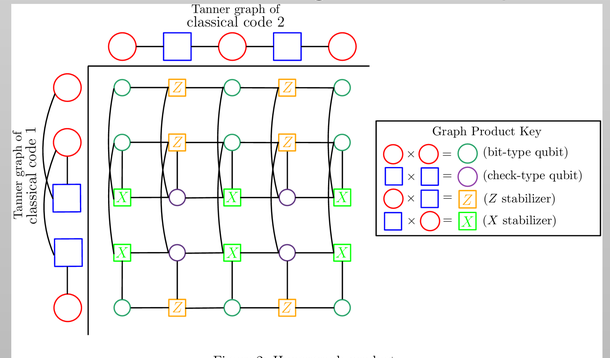

Figure 2: Hypergraph product.

Methods and Current Work

The transformation works by taking linear combinations of parity checks (either X or Z) for each qubit until all support on the qubit is removed, which is equivalent to taking linear combinations of rows in the PCM. The issue with arbitrarily choosing to combine X or Z checks and which way to combine them is that removing one qubit might change the support of a check for a qubit that will be combined in the future, which means we cannot perform the transformation in parallel. Thus, our method works only for codes that can be defined as follows: Given a regular graph $G = (V, E)$, where E represents the qubits (see Fig. 3(a)), we take $G's$ bipartite double-cover to create a regular, bipartite graph where checks are partitioned into sets $V^{(\ell)}$ and $V^{(r)}$ (see Fig. 3(b)). Then, we define a *local code* h for each vertex that acts on the incoming edges (qubits) (see Fig. 3(c)). Since the graph is regular, the same local code can be used for all vertices. Now, the HGP is partitioned into four quadrants, each of which we can independently combine X and Z (see Fig. (4)).This will shift all support into another quadrant, allowing us to remove these qubits.

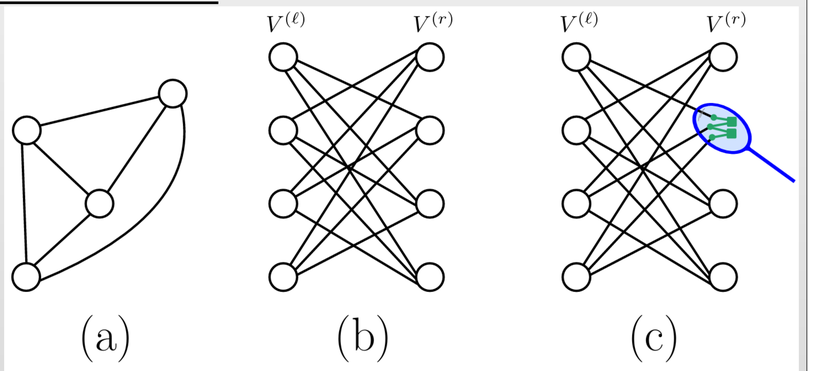

Figure 3: (a) Regular (or complete) graph G; (b) Bipartite double-cover of G; (c) Local code defined for vertices of (b).

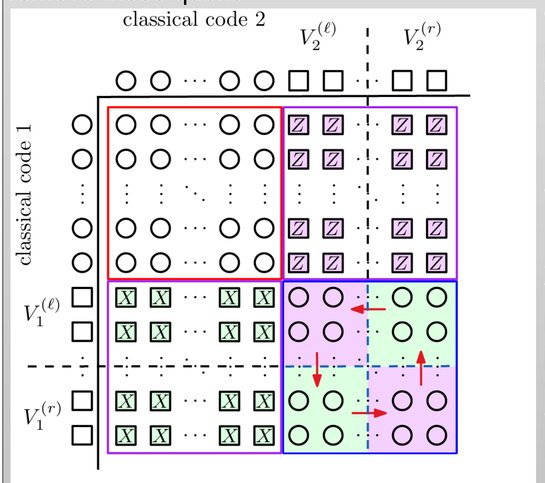

Figure 4: Bipartite classical codes partitioning the qubits into four sets without overlapping support. In the green quadrants, we combine X checks; in the purple quadrants, we combine Z checks, which pushes support in the direction of the red arrows.

We then combine checks as follows: Given a matrix g such that $gh^T = 0$, we can take linear combinations of parity checks in the transpose code as dictated by the rows of g. Since this product is equal to 0, the qubits that come from combining parity checks are no longer used and can be removed. In Fig. 3(c),

$$h = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \Rightarrow h^T = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}. \quad (1)$$

Then,

$$g = (1 \quad 1 \quad 1) \quad (2)$$

satisfies $gh^T \equiv 0 \pmod 2$. Therefore, we combine all three of the checks in each local transpose code. The next goal is to prove the preservation of the code's parameters after the transformation. We are looking at three types of codes: classical repetition codes (see Fig. 5(a)), arbitrary graphs that use single-parity checks for their local codes (see Fig. 5(b)), and arbitrary graphs with arbitrary local codes (see Fig. 5(c)). For the repetition code, we have found that the rank deficiency of the PCM does not change across the transformation, and thus we have used this to prove that the code's useful parameters are maintained under the transformation. We are currently working on the last two cases. The tradeoff of the transformation is that it adds extra weight onto the current checks by combining them. This can lead to adverse effects when it comes to measuring the parities, but sometimes these can be cleverly avoided. We are also attempting to study in which cases these effects can be avoided.

Future Work

By proving that the transformation works nicely for codes like Fig. 4(c), we have partially abstracted the hypergraph product codes that the transformation works on. Could we go further by showing that the transfomation could work in parallel for non-bipartite graphs? Additionally, it would be important to analyze if the transformation works when the classical codes are redundant and show that the parameters stay the same.

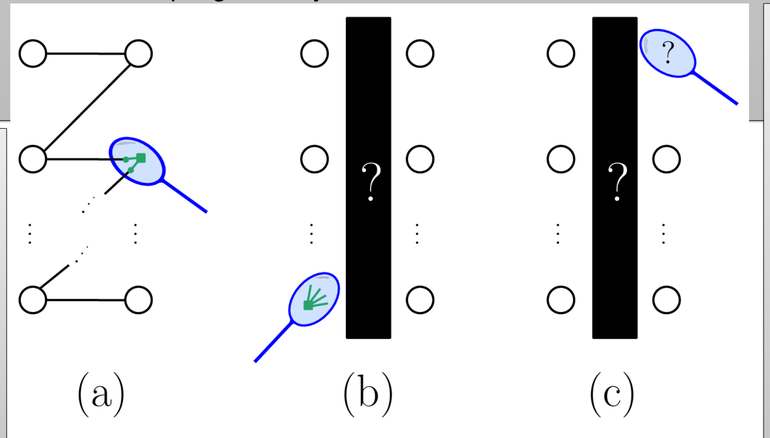

Figure 5: (a) Repetition code; (b) Arbitrary graph with single parity-check local codes; (c) Arbitrary graph with arbitrary local codes.

Acknowledgements and References