A Note on Efficient Clifford Circuit Compilation on CSS codes

Anqi Gong

We consider the task of compiling an arbitrary Clifford circuit/operator (consisting of Hadamard H, phase S, and CNOT gates) acting on bk qubits, which are encoded by b blocks of certain CSS $[\![n,k,d]\!]$ quantum codes, onto a set of native logical gates that could be performed on this CSS code.

The reason why the Pauli gates are omitted is twofold. First, given a circuit mixture of Clifford gates (H, S and CNOT) and Pauli gates (X, Y, Z), one can always pull the Pauli gates to the very end (since conjugation of Pauli gates by Clifford gates still gives Pauli gates with some phase), thus creating a Clifford circuit followed by some Pauli gates. Second, logical Pauli operations are easy to do; for stabilizer codes, they only consist of single-qubit gates and thus are fault-tolerant. Therefore, the actual challenge lies in compiling Clifford gates on a logical level.

The action of a Clifford operator on bk qubits can be described by multiplication with a symplectic matrix [3, 1]. Those matrices form the symplectic group $\mathrm{Sp}(2bk, \mathbb{F}_2)$ under multiplication. This symplectic matrix formalism turns out to be very handy when considering the Clifford synthesis on a logical level [12, 7, 13]. Recently, [10] uses this formalism to prove the *efficiency* of logical Clifford synthesis onto a particular family of codes. We review this symplectic matrix formalism on a logical level in Section 1.

In Section 2, we extract an important message from [10] that holds for general CSS codes: up to certain assumptions, efficiency of logical Clifford synthesis in one block of a certain $[\![n,k,d]\!]$ CSS codes implies the efficiency of the synthesis onto b blocks of such codes. This message is hidden behind the optimization for a specific family of codes in [10].

1 Introduction to symplectic matrix formalism for Clifford gates

Let us spare the precise mathematical definition of symplectic matrix group since it is easier to understand this group from the generator[1] point of view, which are listed in Eq. (1)-(3). What is important is the basis for these matrices in the context of error-correcting codes. Pick an $[\![n,k,d]\!]$ quantum CSS code, let us first consider one block of it, and describe the symplectic group $\mathrm{Sp}(2k, \mathbb{F}_2)$ on k logical qubits. There, a symplectic matrix has $2k$ columns and $2k$ rows. Let $\overline{X}_1, \ldots, \overline{X}_k$ and $\overline{Z}_1, \ldots, \overline{Z}_k$ be a set of X and Z logical representatives of the $[\![n,k,d]\!]$ code. They are paired, i.e., $\overline{X}_i$ only anti-commutes with $\overline{Z}_i$ and commutes with everything else. The $2k$ columns represent $\overline{X}_1, \ldots, \overline{X}_k$ and $\overline{Z}_1, \ldots, \overline{Z}_k$ exactly in this order. We say that the first/last block (first/last k columns) represents the X/Z-logical operators, respectively. It is also important to say that symplectic matrices are invertible, i.e., they describe some invertible logical actions. The symplectic group on bk qubits is thus a subgroup of the special linear group $\mathrm{SL}(2bk, \mathbb{F}_2)$ consisting of all the binary $2bk \times 2bk$ invertible matrices (of determinant one, but this is implied by being invertible, since all the operations are over $\mathbb{F}_2$).

Throughout, we use *square brackets* for symplectic matrices, and *round brackets* for equations that hold for any binary square matrices (may not be symplectic). For example, we will use round brackets when we restrict to the X-logical part (the upper left quadrant) of a symplectic matrix.

When more blocks are involved, we still follow the convention that the first half of the columns are for X-logical operators. To be more precise, the first bk columns are ordered as the k X-logical operators from the first block, then from the second block, etc. Similarly, the next bk columns are for the Z-logical operators. It is important to emphasize that the columns are still paired, i.e., the i-th column from the first block of bk columns and the i-th column from the next block represent a logical X and Z pair. This pairing has the advantage that, to describe a logical action that does not mix X and Z logicals/stabilizers, i.e., the CNOT-type gates in Eq. (1), it is enough to restrict ourselves to the upper left quadrant that concerns the transformation of the X-logicals. The lower right quadrant that describes the transformation of the Z-logicals automatically behaves as the inverse transpose.

$$\text{CNOT-type:} \quad \left[\begin{array}{c|c} U & 0 \\ \hline 0 & U^{-T} \end{array}\right] \in \mathrm{Sp}(2bk, \mathbb{F}_2) \text{ where } U \in \mathrm{SL}(bk, \mathbb{F}_2),\ U^{-T} := (U^T)^{-1} = (U^{-1})^T. \tag{1}$$

$$\text{Phase-type:} \quad \left[\begin{array}{c|c} I & B \\ \hline 0 & I \end{array}\right] \in \mathrm{Sp}(2bk, \mathbb{F}_2) \text{ where } B = B^T \text{ of shape } bk \times bk. \tag{2}$$

$$\text{Hadamard:} \quad \left[\begin{array}{c|c} 0 & I \\ \hline I & 0 \end{array}\right] \in \mathrm{Sp}(2bk, \mathbb{F}_2) \tag{3}$$

Let us give a simple example of CNOT-type gate. Consider two blocks of the same CSS $[\![n,k,d]\!]$ codes. It is known that a transversal CNOT gate between them implements a logical CNOT gate [17]. Say the control is on the first block and target on the second block. The logical operators transform as follows, call $\overline{X}_{1,i}$ and $\overline{X}_{2,i}$ the logical representatives of the first and second block, then $\overline{X}_{1,i} \mapsto \overline{X}_{1,i}\overline{X}_{2,i}$, $\overline{X}_{2,i} \mapsto \overline{X}_{2,i}$,

[1]The fourth type of generator in [12, Table 1] is not necessary for *generation* purpose as proved in [7, Appendix A.A], see also Eq. (16) and Eq. (17). However, these generators, e.g., logical Hadamard on a single code-block, could play a role in the *efficiency* of the synthesis.

and $\overline{Z}_{1,i} \mapsto \overline{Z}_{1,i}$, $\overline{Z}_{2,i} \mapsto \overline{Z}_{1,i}\overline{Z}_{2,i}$. This induced logical action can be described by the following symplectic matrix:

$$\left[\begin{array}{cc|cc} I & I & 0 & 0 \\ 0 & I & 0 & 0 \\ \hline 0 & 0 & I & 0 \\ 0 & 0 & I & I \end{array}\right] \in \mathrm{Sp}(4k, \mathbb{F}_2), \text{ shortened as (taking upper left quadrant) } \left(\begin{array}{c|c} I & I \\ \hline 0 & I \end{array}\right) \in \mathrm{SL}(2k, \mathbb{F}_2). \tag{4}$$

The transpose of it is the transversal CNOT gate with control and target block switched. This is an example of a *cross-block CNOT gate*, which can be defined more generally for two blocks as follows:

$$\text{Cross-block CNOT: } \left(\begin{array}{c|c} I & X \\ \hline 0 & I \end{array}\right) \text{ or } \left(\begin{array}{c|c} I & 0 \\ \hline X & I \end{array}\right), X \text{ arbitrary binary } k \times k \text{ matrix.} \tag{5}$$

In the two-block setting (or later multi-block), one can define an *in-block CNOT gate* where only a single block undergoes logical action (again, X and Z are not mixed, and we restrict ourselves to the upper left quadrant)

$$\text{In-block CNOT on the first block: } \left(\begin{array}{c|c} U & 0 \\ \hline 0 & I \end{array}\right), \text{ on the second block: } \left(\begin{array}{c|c} I & 0 \\ \hline 0 & U \end{array}\right), U \in \mathrm{SL}(k, \mathbb{F}_2). \tag{6}$$

A prominent example of an in-block CNOT gate is automorphism permutations [8], which transforms a stabilizer to a linear combination of other stabilizers, and transforms a logical operator to a linear combination of other logical operators. We restrict ourselves to automorphism permutations that do not give logical identity. As an example, consider the $[\![4, 2, 2]\!]$ code, where $\overline{X}_1 = X_2X_4, \overline{X}_2 = X_3X_4, \overline{Z}_1 = Z_3Z_4, \overline{Z}_2 = Z_2Z_4$ and stabilizers are $S_X = X_1X_2X_3X_4$ and $S_Z = Z_1Z_2Z_3Z_4$. The permutation $i \mapsto i+1 \bmod 4$ clearly leaves the stabilizers invariant, but it also leaves the logical operators invariant too[2]: $\overline{X}_1 = X_2X_4 \mapsto X_3X_1 = \overline{X}_1 \cdot S_X$, etc. In contrast, the permutation of exchanging qubit 2 and 3 has the logical effect of exchanging $\overline{X}_1$ and $\overline{X}_2$, and corresponds to $U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ in Eq. (6).

The problem with pure automorphism permutations is that we can only obtain a small subgroup of $\mathrm{SL}(k, \mathbb{F}_2)$. However, [8] observes that interleaving automorphisms with transversal CNOT gates can enlarge the set of in-block CNOT gate that one is able to do. Restricting to the X-logical, the following cross-block CNOT gates can be generated (U corresponds to some in-block CNOT gates that are achievable with automorphism permutations):

$$\left(\begin{array}{c|c} I & U \\ \hline 0 & I \end{array}\right) = \left(\begin{array}{c|c} U & 0 \\ \hline 0 & I \end{array}\right) \left(\begin{array}{c|c} I & I \\ \hline 0 & I \end{array}\right) \left(\begin{array}{c|c} U^{-1} & 0 \\ \hline 0 & I \end{array}\right), U \in \mathrm{SL}(k, \mathbb{F}_2) \tag{7}$$

The cross-block CNOT gates form an abelian group under matrix multiplication. The U, V need not be invertible in the following equation.

$$\left(\begin{array}{c|c} I & U+V \\ \hline 0 & I \end{array}\right) = \left(\begin{array}{c|c} I & U \\ \hline 0 & I \end{array}\right) \left(\begin{array}{c|c} I & V \\ \hline 0 & I \end{array}\right), U, V \text{ arbitrary binary } k \times k \text{ matrices.} \tag{8}$$

A similar equation holds for block lower-triangular matrices.

Sometimes, if the automorphism group of the CSS code is large enough, i.e., we have a large enough set of U, then any binary $k \times k$ matrix can be written as a sum of them (this is the case for [7, 10]). In this way, we can obtain all the possible cross-block CNOT gates in Eq. (5). In that circumstance, *all* CNOT-type gates in Eq. (1) can be generated herein, we defer the discussion on this to the next section.

There are various ways of implementing in-block CNOT-type gates besides automorphism [6, 18]. When considering the efficiency of Clifford synthesis in the next section, we do not care which method is used, as long as it is efficient. Besides CNOT-type gates, we will also assume the following two gates are efficiently implementable in one code block. For certain self-dual CSS codes, transversal Hadamard with certain CNOT-type gates can give a logical Hadamard gate (on all logical qubits) whose symplectic matrix is shown below. Moreover, through fold-transversal gate [2], i.e., applying CZ gates to each paired of qubits mirrored according to a fold-line and S gates to qubits on the fold-line, and certain CNOT-type gates, one can achieve the following CZ-S gate.

$$\text{Hadamard: } \left[\begin{array}{c|c} 0 & I \\ \hline I & 0 \end{array}\right] \in \mathrm{Sp}(2k, \mathbb{F}_2), \text{ CZ-S: } \left[\begin{array}{c|c} I & I \\ \hline 0 & I \end{array}\right] \in \mathrm{Sp}(2k, \mathbb{F}_2) \tag{9}$$

2 Efficiency in one block implies efficiency in more blocks

In this section, we review the main techniques in [10] regarding compiling Clifford operations efficiently onto (several blocks of) CSS $[\![n, k, d]\!]$ codes. Those techniques are hidden behind their optimizations for the SHYPS codes[3]. We thus find it beneficial to extract the ingredients that could work for CSS codes in general.

[10] defines *efficiency of Clifford compilation* in the following way. Compiling arbitrary Clifford operations onto m qubits (or m blocks of codes each encoding one logical qubit) has a worst-case depth of $\mathcal{O}(m/\log_2 m)$ (achievable by [1]) by a counting argument (assuming gates that have disjoint support can be executed in parallel). However, practical algorithms that do not incur a large constant often do it in $\mathcal{O}(m)$. Therefore, they define a Clifford compilation onto b blocks of a $[\![n, k, d]\!]$ code efficient if the *worst-case depth* is $\mathcal{O}(bk)$.

[2] $[\![4, 2, 2]\!]$ is a quantum Reed-Muller code. In the language of [7], it is the QRM(0, 0, 2) code. It is shown there that the affine part in the automorphism permutation [7, Eq. (3)] has no logical effect for the QRM($r-1, r-1, 2r$) family.

[3] Their code is a hypergraph product of two identical simplex codes. A simplex code is a shortened classical Reed-Muller code of order one.

The techniques in [10] actually allow one to prove the following result. Assume the transversal CNOT between arbitrary two blocks can be implemented in $\mathcal{O}(1)$ time/depth, and also that transversal CNOTs acting on different pairs of blocks can be parallelized[4]. For an $[\![n,k,d]\!]$ CSS code, if implementing arbitrary Clifford gates on one block of it can be done in worst-case time $\mathcal{O}(k)$, then compiling arbitrary Clifford on bk logical qubits (encoded by b blocks of it) can be accomplished in worst-case depth $\mathcal{O}(bk)$, conditioned on either $b \geq k$ or single-pair cross-block logical SWAP gates implementable in depth $\mathcal{O}(1)$ (see the fourth point). We break the proof into the following seven steps that bound the worst-case depth of each operation. In the end, everything can be bounded in terms of the worst-case depth for doing basic *in-block* Clifford gates in Eq. (6),(9).

1. Efficiency of in-block CNOT gates implies the efficiency of cross-block CNOT gates (across two blocks). $D_{\text{cross-block,CNOT}} \leq 3D_{\text{in-block,CNOT}} + 2$. The reverse direction is also true, i.e., efficiency of cross-block CNOT gates implies the efficiency of in-block CNOT gates. $D_{\text{in-block,CNOT}} \leq D_{\text{cross-block,CNOT}}+\text{constant}$.

2. Efficiency of in-block CNOT gates implies the efficiency of in-block phase gates. $D_{\text{in-block,phase}} \leq 3D_{\text{in-block,CNOT}} + 2D_{CZ-S}$.

3. Efficiency of cross-block CNOT gates implies the efficiency of cross-block phase gates (across two blocks). $D_{\text{cross-block,phase}} \leq D_{\text{cross-block,CNOT}} + 2D_{\text{Hadamard}}$.

4. For the multi-block logical permutation gate, $D_{\text{multi-block,perm}} \leq D_{\text{in-block,perm}} + 3\cdot k \cdot D_{\text{cross-block,SWAP}}$. One might notice that b does not play a role and instead, k is the factor. An in-block permutation gate is a form of in-block CNOT-type gate, $D_{\text{in-block,perm}} \leq D_{\text{in-block,CNOT}}$. Cross-block SWAP gate here refers to swapping a single logical qubit from one block to another logical qubit from another block (rather than swapping multiple logical qubits between two blocks), it is a form of cross-block CNOT-type gate, so $D_{\text{cross-block,SWAP}} \leq D_{\text{cross-block,CNOT}}$.

5. Efficiency of in-block and cross-block CNOT gates implies the efficiency of multi-block CNOT gates. $D_{\text{multi-block,CNOT}} \leq (2b-1)D_{\text{cross-block,CNOT}} + D_{\text{in-block,CNOT}} + D_{\text{multi-block,perm}}$.

6. Efficiency of in-block and cross-block phase gates implies the efficiency of multi-block phase gates. $D_{\text{multi-block,phase}} \leq (b-1)D_{\text{cross-block,phase}} + D_{\text{in-block,phase}}$.

7. Arbitrary Clifford-gate over b blocks has worst-case depth $D_{\text{multi-block,Clifford}} \leq 3D_{\text{multi-block,phase}} + D_{\text{multi-block,CNOT}} + 2D_{\text{Hadamard}}$.

Before proving these statements, let us give a few remarks. All the steps are sufficient conditions and say nothing about necessity, i.e., the inequalities are not likely to be tight. In fact, there can also be shortcuts to each type of gate for certain codes. The only peculiar condition is the fourth point: There, the multi-block logical permutation is efficient if $b \geq k$ or, dropping this assumption but requiring $D_{\text{cross-block,perm}} \sim \mathcal{O}(1)$. The latter condition happens to hold for the codes in [10].

For the ease of recursion, the number of blocks b is always assumed to be a power of two, e.g., $b = 2^a$.

(1) follows from [10, Lemma IX.7] which states that an arbitrary square matrix M can be written as the sum of at most two invertible matrices U, V. Either Eq. 7 applies (M is itself invertible) or we can decompose the cross-block CNOT gate into two transversal CNOT gates (depth-one each) and three in-block CNOT-type gates as follows (recall that we are restricting ourselves to the upper-left quadrant of the symplectic matrices)

$$\left(\begin{array}{c|c} I & U+V \\ \hline 0 & I \end{array}\right) = \left(\begin{array}{c|c} U & 0 \\ \hline 0 & I \end{array}\right)\left(\begin{array}{c|c} I & I \\ \hline 0 & I \end{array}\right)\left(\begin{array}{c|c} U^{-1}V & 0 \\ \hline 0 & I \end{array}\right)\left(\begin{array}{c|c} I & I \\ \hline 0 & I \end{array}\right)\left(\begin{array}{c|c} V^{-1} & 0 \\ \hline 0 & I \end{array}\right) \tag{10}$$

Let us look at the reverse direction, i.e., generating in-block CNOT gates from the cross-block CNOT gate. For automorphism-based gates, this direction is to fill in the in-block CNOT gates that the automorphism permutations weren't able to do.

[8, Thm. 4] says that any in-block CNOT gates Eq. (6) can be generated, conditioned on all cross-block CNOT gates Eq. (5) can be realized. The unitary process there (the ancilla block undergoes no logical transform in the end) is not likely to be efficient. [10, Lemma IX.18] gives a shortcut to that: to implement an in-block CNOT gate $U \in \mathrm{SL}(k, \mathbb{F}_2)$, apply a CNOT $\left(\begin{array}{c|c} I & U^{-1} \\ \hline 0 & I \end{array}\right)$ across data and ancilla block; measure the data block transversally in the X basis and apply conditional Z correction to the ancilla block.

Here we give an alternative way of doing this, which is conceptually easier to understand (worse constant but enough to serve the efficiency argument) and does not require conditional correction. Specifically, any in-block CNOT gate $U \in \mathrm{SL}(k, \mathbb{F}_2)$ can be generated using three cross-block CNOT gates and three transversal CNOT gates, i.e., $D_{\text{in-block,CNOT}} \leq 3D_{\text{cross-block,CNOT}} + 3$. Implement an in-block CNOT U on the data block and in-block CNOT U^{-1} on the ancilla block, i.e., $\left(\begin{array}{c|c} U & 0 \\ \hline 0 & U^{-1} \end{array}\right) = \left(\begin{array}{c|c} 0 & I \\ \hline I & 0 \end{array}\right)\left(\begin{array}{c|c} 0 & U^{-1} \\ \hline U & 0 \end{array}\right)$ as follows, then discard the ancilla block by measuring it.

$$\left(\begin{array}{c|c} 0 & U^{-1} \\ \hline U & 0 \end{array}\right) = \left(\begin{array}{c|c} I & U^{-1} \\ \hline 0 & I \end{array}\right)\left(\begin{array}{c|c} I & 0 \\ \hline U & I \end{array}\right)\left(\begin{array}{c|c} I & U^{-1} \\ \hline 0 & I \end{array}\right), \quad \left(\begin{array}{c|c} 0 & I \\ \hline I & 0 \end{array}\right) = \left(\begin{array}{c|c} I & I \\ \hline 0 & I \end{array}\right)\left(\begin{array}{c|c} I & 0 \\ \hline I & I \end{array}\right)\left(\begin{array}{c|c} I & I \\ \hline 0 & I \end{array}\right). \tag{11}$$

For (2), we again give an alternative treatment than [10] since theirs is code-specific. To realize any in-block phase gate $\left[\begin{array}{c|c} I & B \\ \hline 0 & I \end{array}\right] \in \mathrm{Sp}(2k, \mathbb{F}_2)$ efficiently, where $B = B^T$, we decompose it into in-block CNOT gates and CZ-S gates using the following lemma (proof in Appendix A) and the next two equations.

[4]This assumption is fair: compared to compilation onto bk blocks of codes each encoding a single logical qubit, we need less all-to-all *transversal* connectivity for b blocks of $[\![n,k,d]\!]$ codes.

Lemma 1. *For $k > 2$, any binary symmetric $k \times k$ matrix B can be written as either $B = UU^T$ or $B = UU^T + VV^T$ for invertible matrices U, V.*

$$\left[\begin{array}{c|c} I & UU^T \\ \hline 0 & I \end{array}\right] = \left[\begin{array}{c|c} U & 0 \\ \hline 0 & U^{-T} \end{array}\right] \left[\begin{array}{c|c} I & I \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} U^{-1} & 0 \\ \hline 0 & U^T \end{array}\right],$$

$$\left[\begin{array}{c|c} I & UU^T + VV^T \\ \hline 0 & I \end{array}\right] = \left[\begin{array}{c|c} U & 0 \\ \hline 0 & U^{-T} \end{array}\right] \left[\begin{array}{c|c} I & I \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} U^{-1}V & 0 \\ \hline 0 & U^T V^{-T} \end{array}\right] \left[\begin{array}{c|c} I & I \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} V^{-1} & 0 \\ \hline 0 & V^T \end{array}\right].$$

For (3), we can decompose a cross-block phase gate into a CNOT-type gate conjugated by Hadamard on the second block [10, Proof of Lemma X.20]:

$$\left[\begin{array}{cc|cc} I & 0 & 0 & A \\ 0 & I & A^T & 0 \\ \hline \multicolumn{2}{c|}{\multirow{2}{*}{0}} & I & 0 \\ & & 0 & I \end{array}\right] = \left[\begin{array}{cc|cc} I & 0 & 0 & 0 \\ 0 & 0 & 0 & I \\ \hline 0 & 0 & I & 0 \\ 0 & I & 0 & 0 \end{array}\right] \left[\begin{array}{cc|cc} I & A & \multicolumn{2}{c}{\multirow{2}{*}{0}} \\ 0 & I & & \\ \hline \multicolumn{2}{c|}{\multirow{2}{*}{0}} & I & 0 \\ & & A^T & I \end{array}\right] \left[\begin{array}{cc|cc} I & 0 & 0 & 0 \\ 0 & 0 & 0 & I \\ \hline 0 & 0 & I & 0 \\ 0 & I & 0 & 0 \end{array}\right] \tag{12}$$

(4) is the most peculiar step [10, Thm. XI.4]. The worst-case depth being independent of b is a consequence of the fact that a permutation can be written as the product of *two* involutions[5], while the k factor[6] comes from multi-graph edge-coloring (Vizing's theorem, inspired by Shannon [15]).

To be more precise, let us write $P = A \cdot B$, where A and B are involutions, that is to say, each of them only contains SWAPs on disjoint pairs. Therefore, we can split those SWAPs into those whose support (a pair of logical qubits) are in the same block A_{IB}, B_{IB}, or those across two blocks A_{CB}, B_{CB}, and write $P = A_{CB}A_{IB}B_{IB}B_{CB}$. $A_{IB}B_{IB}$ is an in-block permutation gate and has worst-case depth $D_{\text{in-block,perm}}$. For the cross-block SWAP part, construct a multi-graph with b vertices, each correspond to a code-block, and add an edge for each cross-block SWAP. Color the edges of this multi-graph such that no two edges sharing one (or more) common vertices have the same color. Edges (SWAP gates) that have the same color can be scheduled to execute in parallel. The maximum degree of this multi-graph is $\leq k$ and $\lceil \frac{3}{2}k \rceil$ colors suffice [15][7].

For (5,6), the treatments for efficient multi-block CNOT-type gates [10, Lemma IX.20] and phase-type gates [10, Lemma X.22] are similar. They recursively divide the compilation into two disjoint sets of $b/2$ blocks each. Handle things inside the two disjoint sets in parallel, and use a cyclic scheduling for operations across the two sets. Let $b = 2^a$ for the ease of recursion.

Let us explain (6) first since the phase-type gates $\left[\begin{array}{c|c} I & G \\ \hline 0 & I \end{array}\right] \in \mathrm{Sp}(2bk, \mathbb{F}_2)$ (G symmetric) commute with each other and are easier to deal with. Here we illustrate the idea using $b = 4$. There the symmetric matrix G from the upper right quadrant is in the form $G = \begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ A_{1,2}^T & A_{2,2} & A_{2,3} & A_{2,4} \\ A_{1,3}^T & A_{2,3}^T & A_{3,3} & A_{3,4} \\ A_{1,4}^T & A_{2,4}^T & A_{3,4}^T & A_{4,4} \end{pmatrix}$. $A_{i,i}$ are symmetric matrices and they correspond to in-block phase-type gates on the four blocks respectively, so they can be applied in parallel in depth $D_{\text{in-block,phase}}$. For the remaining, one can divide the four blocks into two disjoint set. Do the cross-block phase gate in the two sets in parallel. Each set shall be further recursively decomposed, but in the $b = 4$ case, no further decomposition is needed and we can just do $A_{1,2}, A_{3,4}$ (each using Eq. (12)) in parallel. For the gates across the two sets, no recursion is needed, just handle them via cyclic scheduling, i.e., parallelizing gates on the same secondary diagonals (those parallel to the main diagonal). If each set contains $b/2$ blocks, then depth $\frac{b}{2}D_{\text{cross-block,phase}}$ suffices. In the $b = 4$ case, for $\begin{pmatrix} A_{1,3} & A_{1,4} \\ A_{2,3} & A_{2,4} \end{pmatrix}$, do $A_{1,3}, A_{2,4}$ in parallel, then do $A_{1,4}, A_{2,3}$ in parallel. Combining these procedures, we obtain a $(b-1)$ round scheduling for any cross-block phase gates on b blocks.

For CNOT-type gates on b blocks in Eq.(1), we will apply the block PLDU decomposition recursively to the upper left quadrant $U \in \mathrm{SL}(bk, \mathbb{F}_2)$ describing the bk X-logical operators, again $b = 2^a$.

Recall the block PLDU decomposition[8] for any binary invertible matrix $X \in \mathrm{SL}(2k', \mathbb{F}_2)$:

$$X = P \left(\begin{array}{c|c} I & 0 \\ \hline L & I \end{array}\right) \left(\begin{array}{c|c} C_1 & 0 \\ \hline 0 & C_2 \end{array}\right) \left(\begin{array}{c|c} I & U \\ \hline 0 & I \end{array}\right), \tag{13}$$

where P is a permutation matrix, L and U are binary matrices of shape $k' \times k'$, and $C_1, C_2 \in \mathrm{SL}(k', \mathbb{F}_2)$. We can further decompose C_1, C_2 into PLDU forms. In particular, we can pull the permutation parts P_1, P_2 from their decompositions to the front

$$\left(\begin{array}{c|c} I & 0 \\ \hline L & I \end{array}\right) \left(\begin{array}{c|c} P_1 & 0 \\ \hline 0 & P_2 \end{array}\right) = \left(\begin{array}{c|c} P_1 & 0 \\ \hline 0 & P_2 \end{array}\right) \left(\begin{array}{c|c} I & 0 \\ \hline P_2^{-1}LP_1 & I \end{array}\right). \tag{14}$$

Merging P and $\left(\begin{array}{c|c} P_1 & 0 \\ \hline 0 & P_2 \end{array}\right)$ again leads to a permutation matrix. Continuing this process further, we get the following lemma:

[5]An involution layer can be understood as a series of SWAPs on disjoint support that can take place in parallel. One can prove this by decomposing any permutation into disjoint cycles, then write every even or odd cycle as a product of two involutions following Exercise 10.1.17 of [14]: $(1, \ldots, 2n) = [(n, n+1), (n-1, n+2) \cdots (1, 2n)] \cdot [(n, n+2)(n-1, n+3) \cdots (2, 2n)]$ and $(1, \ldots, 2n+1) = [(n, n+1)(n-1, n+2) \cdots (1, 2n)] \cdot [(n, n+2)(n-1, n+3) \cdots (1, 2n+1)]$.

[6]Consider the adversarial example given in [11] that we need to compile the following: there exists a two-qubit gate from qubit i of the first block to some other qubit in the i-th block for each $i \in [k]$. We cannot avoid a factor of k in that case.

[7]This number is tighter than Vizing's theorem (maximum degree plus multiplicity and hence $2k$ colors) in this case.

[8]For invertible X, we can first apply a permutation matrix P, such that $P^{-1}X$ has an invertible upper-left quadrant A, then use the block LDU decomposition $\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) = \left(\begin{array}{c|c} I & 0 \\ \hline CA^{-1} & I \end{array}\right) \left(\begin{array}{c|c} A & 0 \\ \hline 0 & D - CA^{-1}B \end{array}\right) \left(\begin{array}{c|c} I & A^{-1}B \\ \hline 0 & I \end{array}\right)$.

Lemma 2. *Any invertible matrix $X \in SL(2^a k, \mathbb{F}_2)$ can be written as follows*

$$X = P \left(\begin{array}{c|c} I & 0 \\ \hline L_0 & I \end{array}\right) \left(\begin{array}{c|c} \begin{smallmatrix} I & 0 \\ L_{1,1} & I \end{smallmatrix} & 0 \\ \hline 0 & \begin{smallmatrix} I & 0 \\ L_{1,2} & I \end{smallmatrix} \end{array}\right) \cdots \begin{pmatrix} C_1 & & & \\ & C_2 & & \\ & & \ddots & \\ & & & C_{2^a} \end{pmatrix} \cdots \left(\begin{array}{c|c} \begin{smallmatrix} I & U_{1,1} \\ 0 & I \end{smallmatrix} & 0 \\ \hline 0 & \begin{smallmatrix} I & U_{1,2} \\ 0 & I \end{smallmatrix} \end{array}\right) \left(\begin{array}{c|c} I & U_0 \\ \hline 0 & I \end{array}\right), \tag{15}$$

where $L_{i,j}, U_{i,j}, j \in [2^i]$ are binary square matrices with $2^{a-i-1}k$ rows/columns, and $C_i \in SL(k, \mathbb{F}_2), i \in [2^a]$.

Now, for the lower-triangular matrices in Eq. (15), we can again use the cyclic scheduling for each of them[9]. For example, we need depth $2^{a-1} \cdot D_{\text{cross-block,CNOT}}$ for L_0, and depth $2^{a-2} \cdot D_{\text{cross-block,CNOT}}$ for cyclically going through $L_{1,1}, L_{1,2}$ in parallel, etc. Same argument for the upper-triangular matrices, together with the worst-case depth for the permutation matrix P and the in-block CNOT cost of the middle block-diagonal matrix, the total depth amounts to $(2b-1) \cdot D_{\text{cross-block,CNOT}} + D_{\text{in-block,CNOT}} + D_{\text{multi-block,perm}}$.

Deriving (7) from (5,6) follows from the following decomposition of an arbitrary Clifford operator (modulo logical Pauli) $\chi \in \text{Sp}(2bk, \mathbb{F}_2)$ [10, Thm. XII.1]:

$$\chi = \left[\begin{array}{c|c} I & K \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} I & 0 \\ \hline E & I \end{array}\right] \left[\begin{array}{c|c} F & 0 \\ \hline 0 & F^{-T} \end{array}\right] \left[\begin{array}{c|c} I & G \\ \hline 0 & I \end{array}\right], \tag{16}$$

where F invertible and K, E, G are symmetric.

Note that

$$\left[\begin{array}{c|c} I & 0 \\ \hline E & I \end{array}\right] = \left[\begin{array}{c|c} 0 & I \\ \hline I & 0 \end{array}\right] \left[\begin{array}{c|c} I & E \\ \hline 0 & I \end{array}\right] \left[\begin{array}{c|c} 0 & I \\ \hline I & 0 \end{array}\right], \tag{17}$$

where $\left[\begin{array}{c|c} 0 & I \\ \hline I & 0 \end{array}\right]$ corresponds to every block applying an in-block Hadamard gate to itself in parallel.

3 Discussions

One should be more specific about how *depth* is defined. For example, in [10], they are essentially[10] defining the implementation of any automorphism permutation to be a single-depth operation. They also assume this can be done in $\mathcal{O}(1)$ *time*, even though the permutation σ comes from an automorphism group as large as $\text{SL}(r, \mathbb{F}_2) \times \text{SL}(r, \mathbb{F}_2)$ (the size of this group is roughly 2^{2r^2-4}, where $r = \sqrt{k} \sim \log \sqrt{n}$). While this may be possible for the photonics platform [16], $\mathcal{O}(1)$ time is a quite challenging assumption for many other platforms where long-range connectivity is possible but limited, as implementing *arbitary* σ will incur some hidden time cost[11] that scales with n.

In our framework, we absorb the hidden time cost into in-block gate depth, as besides automorphism-based gates, there are other alternatives like surgery-type gates [4], selective measurement gadgets [18], or gate teleportation [11] that might reduce the time to $\mathcal{O}(1)$ by trading resources and space.

Another thing is that [10] defines efficiency for worst-case Clifford operators. It makes sense since the native gate sets could be vastly different for $k = 1$ and $k > 1$ codes, e.g., for $k > 1$ codes, single logical qubit addressing could be hard while some collective CNOT gates using automorphism are easy. However, practical algorithms are probably not requiring worst-case Clifford in between non-Clifford gates. In that case, can $k > 1$ codes still be as efficient as $k = 1$ codes in time?

References

[1] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5), November 2004.

[2] Nikolas P. Breuckmann and Simon Burton. Fold-transversal Clifford gates for quantum codes. *Quantum*, 8:1372, June 2024.

[3] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane. Quantum error correction and orthogonal geometry. *Phys. Rev. Lett.*, 78:405–408, Jan 1997.

[4] Lawrence Z. Cohen, Isaac H. Kim, Stephen D. Bartlett, and Benjamin J. Brown. Low-overhead fault-tolerant quantum computing using long-range connectivity. *Science Advances*, 8(20), May 2022.

[5] Nathan Constantinides, Ali Fahimniya, Dhruv Devulapalli, Dolev Bluvstein, Michael J. Gullans, J. V. Porto, Andrew M. Childs, and Alexey V. Gorshkov. Optimal routing protocols for reconfigurable atom arrays, 2024.

[6] Andrew Cross, Zhiyang He, Patrick Rall, and Theodore Yoder. Improved QLDPC surgery: Logical measurements and bridging codes, 2024.

[7] Anqi Gong and Joseph M. Renes. Computation with quantum reed-muller codes and their mapping onto 2d atom arrays, 2024.

[9] This uses the fact that multiplication in $\left\{\left(\begin{array}{c|c} I & 0 \\ \hline L_0 & I \end{array}\right)\right\}$ is commutative, same holds for $\left\{\left(\begin{array}{c|c} \begin{smallmatrix} I & 0 \\ L_{1,1} & I \end{smallmatrix} & 0 \\ \hline 0 & \begin{smallmatrix} I & 0 \\ L_{1,2} & I \end{smallmatrix} \end{array}\right)\right\}$, etc.

[10] To be more precise, [10] assumes that for two blocks of the $[\![n, k, d]\!]$ codes (qubits are indexed from 1 to n and $n+1$ to $2n$ for the two blocks), for *any* permutations σ on the n qubits that comes from a set of size exponentially large in k, $\Pi_{i=1}^{n} \text{CNOT}_{i,n+\sigma(i)}$ can be implemented in $\mathcal{O}(1)$. One can break such a parallel CNOT gate into three steps, doing σ on the second block, applying transversal CNOT, and undoing σ on the second block.

[11] With the hypercube connectivity like in the neutral atom platform, using the 1D routing protocol from [5] (no need for the proposed trap upgrade), each σ for the codes in [10] can be implemented in time $\mathcal{O}(r)$ – route the two axes separately and treat each problem as a 1D routing problem for $\text{SL}(r, \mathbb{F}_2)$.

[8] Markus Grassl and Martin Roetteler. Leveraging automorphisms of quantum codes for fault-tolerant quantum computation. In *2013 IEEE International Symposium on Information Theory*. IEEE, July 2013.

[9] Abraham Lempel. Matrix factorization over $GF(2)$ and trace-orthogonal bases of $GF(2^n)$. *SIAM J. Comput.*, 4(2):175–186, June 1975.

[10] Alexander J Malcolm, Andrew N Glaudell, Patricio Fuentes, Daryus Chandra, Alexis Schotte, Colby DeLisle, Rafael Haenel, Amir Ebrahimi, Joschka Roffe, Armanda O Quintavalle, et al. Computing efficiently in qldpc codes. *arXiv preprint arXiv:2502.07150*, 2025.

[11] Quynh T. Nguyen and Christopher A. Pattison. Quantum fault tolerance with constant-space and logarithmic-time overheads, 2024.

[12] Narayanan Rengaswamy, Robert Calderbank, Henry D. Pfister, and Swanand Kadhe. Synthesis of logical clifford operators via symplectic geometry. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 791–795, 2018.

[13] Hasan Sayginel, Stergios Koutsioumpas, Mark Webster, Abhishek Rajput, and Dan E Browne. Fault-tolerant logical clifford gates from code automorphisms, 2025.

[14] William R. Scott. *Group theory*. Prentice-Hall, 1964.

[15] Claude E. Shannon. *A Theorem on Coloring the Lines of a Network*, pages 584–587. 1993.

[16] Stephanie Simmons. Scalable fault-tolerant quantum technologies with silicon color centers. *PRX Quantum*, 5:010102, Mar 2024.

[17] Andrew M. Steane. Efficient fault-tolerant quantum computing. *Nature*, 399(6732):124–126, May 1999.

[18] Qian Xu, Hengyun Zhou, Guo Zheng, Dolev Bluvstein, J. Pablo Bonilla Ataides, Mikhail D. Lukin, and Liang Jiang. Fast and parallelizable logical computation with homological product codes, 2024.

A Proof of Lemma 1

We now prove Lemma 1 that says for $k > 2$, any binary symmetric $k \times k$ matrix B can be written as either $B = UU^T$ or $B = UU^T + VV^T$ for invertible matrices U, V.

We will be mainly using techniques from [9]. In particular, [9] proves that if B is a full-rank binary symmetric matrix with a non-zero diagonal, then it can be written as UU^T for invertible U. For rank-deficient B, [9] says that one can find a permutation matrix P and a full-rank $T = \begin{pmatrix} I & 0 \\ S & I \end{pmatrix}$ such that $B = P^T T \begin{pmatrix} L & 0 \\ 0 & 0 \end{pmatrix} T^T P$ and L is a square full-rank symmetric matrix.

Let us first show that one can always write an $m \times m$ identity matrix I_m as the sum of two binary symmetric full-rank matrices for $m \geq 2$. We do this via induction.

$$I_2 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \; I_3 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}. \tag{18}$$

We name the two matrices appearing in the I_m decomposition $M_{m,1}$ and $M_{m,2}$. Any integer $m \geq 4$ can be written as $2k_1 + 3k_2$ for some integers k_1, k_2. We can obtain the decomposition of I_m as

$$M_{m,1} = \left(\oplus_{i=1}^{k_1} M_{2,1}\right) \oplus \left(\oplus_{i=1}^{k_2} M_{3,1}\right), \; M_{m,2} = \left(\oplus_{i=1}^{k_1} M_{2,2}\right) \oplus \left(\oplus_{i=1}^{k_2} M_{3,2}\right). \tag{19}$$

It is clear that $M_{m,1}$ and $M_{m,2}$ are also symmetric and full-rank.

L being a square full-rank symmetric matrix means that either (1) we can find an invertible matrix S such that $L = SS^T$ (if L has a non-zero diagonal), or (2) L has an even number of rows/columns (say $2k'$) and one can find an invertible matrix S such that $L = S \begin{pmatrix} 0 & I_{k'} \\ I_{k'} & 0 \end{pmatrix} S^T$.

For the first possibility, say L has k'' rows/columns, then

$$B = P^T T \left[\begin{pmatrix} SM_{k'',1}S^T & 0 \\ 0 & I_{k-k''} \end{pmatrix} + \begin{pmatrix} SM_{k'',2}S^T & 0 \\ 0 & I_{k-k''} \end{pmatrix} \right] T^T P \tag{20}$$

works because $M_{k'',1}$ and $M_{k'',2}$ are symmetric and full-rank.

For the second possibility:

$$B = P^T T \left[\begin{pmatrix} S \begin{pmatrix} I_{k'} & 0 \\ 0 & M_{k',1} \end{pmatrix} S^T & 0 \\ 0 & I_{k-2k'} \end{pmatrix} + \begin{pmatrix} S \begin{pmatrix} I_{k'} & I_{k'} \\ I_{k'} & M_{k',1} \end{pmatrix} S^T & 0 \\ 0 & I_{k-2k'} \end{pmatrix} \right] T^T P. \tag{21}$$

In particular, $\begin{pmatrix} I_{k'} & I_{k'} \\ I_{k'} & M_{k',1} \end{pmatrix}$ is full-rank, as through row operations, it becomes $\begin{pmatrix} I_{k'} & I_{k'} \\ 0_{k'} & I_{k'} + M_{k',1} \end{pmatrix} = \begin{pmatrix} I_{k'} & I_{k'} \\ 0_{k'} & M_{k',2} \end{pmatrix}$ and $M_{k',2}$ is invertible.