Optimize Surface Code Braiding

Fei Hua
Rutgers Department of Computer Science
Rutgers University
New Jersey, 08817
Email:huafei90@gmail.com

eddy z. zhang
Rutgers Department of Computer Science
Rutgers University
New Jersey, 08817
Email:eddy.zhengzhang@gmail.com

***Abstract*—There has been rapid development in quantum computation technology in recent years. However, the physical qubits are fickle and subject to errors. Surface code is a highly reliable error correction code. Scheduling braiding and mapping qubits in surface code circuits remains a challenging and yet important problem. In this paper, we propose and analyze different types of automatic braiding and mapping techniques in order to minimizing the circuits equipped with surface codes.**

I. Background

Recently quantum computer has become an appealing alternative for solving particular problems which can not be solved in reasonable amount of time using a classical computer. For example factoring a big number problem is generally thought to have exponential time cost on a classical computer [1]. However efficient algorithms take polynomial time on a quantum computer using Shor's algorithm [2].

Current quantum computer implementations are prone to noises, they need to be equipped with error correction mechanism to be practically useful. One approach to build a quantum computer is based on Surface codes [3]. Surface codes are a family of promising, low-overhead quantum error correction (QEC) codes [4], where logical qubit information is encoded in the topology of a two-dimensional lattice of physical qubits [5]. There are two main avors of surface codes: planar [6] and double-defect [27] encodings. In both cases, a logical qubit is encoded using multiple physical qubits.

Braiding [3] is a technique for implementing CNOT gate in surface code. We show an example in Fig. 1. Each filled blue circle in the 2D grid represents a logic qubit. To perform braiding, one logical qubit needs to find a path to another logical qubit, which are illustrated as solid lines within the graid in Fig. 1. Concurrent braiding routes can not have crossings. Two braiding operations that cross each other can only run sequentially. How to schedule braiding automatically and efficiently is an active research area.

A number of works have considered how to map from logical qubits to physical qubits. However, most of the work is on circuits that do not have quantum error correction codes (?? cite). A lot of work still has to be done for the qubit mapping problem in circuits equipped with surface codes. Good mapping may enable concurrent CNOT operations as much as possible and improve the parallelism. Bad mapping may introduce many braiding crossings and deteriorate the performance.

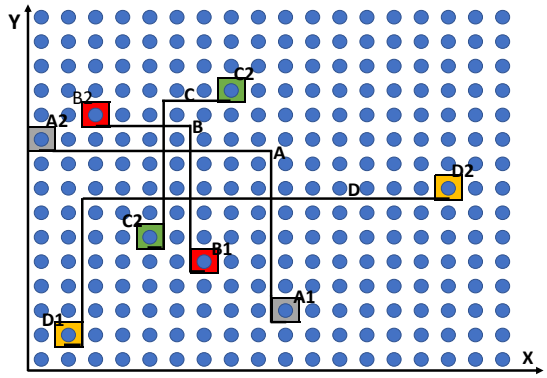

Fig. 1: An example of braiding route, the blue circle is the logic qubit in surface code, the route is the braiding of CNOT operation

II. Motivation

A. Recent work

The work by Javadi-Abhari et al. work [7] explores the method of list scheduling and aims to schedule as many operations in a cycle as possible. However, to obtain an initial mapping, they aim to minimize the distance between different tiles, which is only a coarse estimate of the number of crossings for braiding operations. Other works [8] only consider dependence caused serialization but not braiding-crossing caused serialization. Other works model the braiding problem in the 3D space by adding time as a new dimension [9] [10] [11]. However, these techniques are all manual. There is a lack of systematic exploration for formal and automatic methods to obtain optimal initial mapping and exact braiding schedule for any given circuits.

B. Braiding route change

Different braiding routes for a pair of logical qubits can run in the same number of surface code cycles as summarized in Austin's work [3], since the execution of a braiding is determined by the distance d which is the distance between two defects in the double-defect encoding. The total number of cycles for a braiding is $d + 1$ since a braiding contains two movement. No matter how long the braiding paths are,

as long they are topologically equivalent, the time cost is the same.

III. Methods

A. Define braiding route and crossing

First we define the braiding route and crossing. Suppose we have two qubits D1 with (x, y) coordinates (1,1) and D2 with coordinates (15,7) as shown in Fig. 1. We define an unique routing path m for any braiding as follows. The route m starts from the node D1, move vertically to the the coordinates (D1.x,D2.y), then it moves horizontally to (D2.x, D2.y). Here suppose we have logical qubits A1, A2, B1, B2 and operations CNOT(A1,A2), CNOT(B1,B2), we can define unique braiding paths using the definition above. We can also check whether there is a crossing between two CNOT using unique definition of the braiding paths. For simplicity of illustration, we let axs = min(A1.x, A2.x), axm = max(A1.x,A2.x),ays = min(A1.y,A2.y), aym = max(A1.y,A2.y), bxs = min(B1.x,B2.x), bxm = max(B1.x,B2.x),bys = min(B1.y,B2.y), bym = max(B1.y,B2.y)

$$\begin{aligned} &bxs > axm || bys > aym || bxm < axs || bym < ays || \\ &axs > bxs \&\& axm < bxm \&\& ays > bys \&\& aym < bym || \\ &axs < bxs \&\& axm > bxm \&\& ays < bys \&\& aym > bym \end{aligned} \tag{1}$$

If the result is true, then we say there is no crossing between two CNOT operations, otherwise there is a crossing.

B. Find initial mapping

There are many related works that deal with the initial mapping [12], but none of them accounts for braiding crossings in surface code circuits. Based on the previously defined crossings and route, we w define the initial mapping L with minimum cost function as follows: .

$$Initial\ L = (\forall L \in LS) min(\sum_{i=1}^{n} C(L, oper_i))/2 \tag{2}$$

$\mathbf{L}$ is a mapping in the set $\mathbf{LS}$ which contains all possible mappings between physical qubits and logical qubits, $\mathbf{oper_i}$ is the i-th quantum CNOT operation. $\mathbf{C(L, oper_i)}$ means given a mapping L, the number of crossings caused by i-th CNOT operation. We want choose the mapping $\mathbf{L}$ with the minimum number of crossings.

C. Schedule all operations

We want to introduce an execution set $\mathbf{e_{ij}}$.

$\mathbf{S}$ is a set contains all operations on circuit and $\mathbf{s_i}$ is a subset of S. s_i contains all operations which do not have any qubit dependency. The relation constraint between s_i and s_{i+1} is that s_{i+1} is executed after s_i which does not violate the qubit dependency. There are multiple ways to split S to s_i, for instance, we have x,y,z ways to split S. And the cost for each split method is different: Cost(x), Cost(y), Cost(z).

$\mathbf{e_{ij}}$ is the execution set and is a subset of s_i. e_{ij} contains operations that do not have any crossing or dependency. So all operations in e_{ij} can run in parallel. Also we know that there are multiple ways to divide s_i into e_{ij}, different methods may have different cost. $e_{i+1,j}$ is the subset of S_{i+1} and $e_{i,j+1}$ is executed after $e_{i,j}$ which does not violate operation crossings. We define the time cost of e_{ij} as one unit time as all operations in e_{ij} can truly run in parallel. Given the way to divide S to s_i and to e_{ij} the total execution time cost is:

$$Total\ Cost = \sum_{i=1}^{k} S_i = \sum_{i=1}^{k} \sum_{j=1}^{|S_k|} e_{ij}. \tag{3}$$

There are multiple ways to divided S to $\mathbf{s_i}$ and $\mathbf{e_ij}$ sets, but we want to find a way which has the minimum total cost. For example based on Fig. 1, if we have A,B,C,D four operations with logic qubits A1,A2,B1,B2,C1,C2,D1,D2 Then we draw routes of four operations. No operation have any qubit dependency, so the ready set s_1 is A,B,C,D, However we can find no operations that can run in parallel since they have crossings with each other using the braiding path in Fig. 1. So we need to divide the set s_1 into four execution sets $\{e_{11}\}, \{e_{12}\}, \{e_{13}\}.\{e_{14}\}$ which needs to execute in sequential order for four times. However we have three options here that may lead to different braiding solutions and potentially minimize the execution time.

1) Change mapping by swapping

Based on Fig. 1, if we swap the states of c1 and b2, d1 and a1 using operation swap(c1,b2) and swap(d2,a1) on figure 2, then all operations do not have any crossing, then we only need to divide the set s_1 to one execution set $\{e_{11}\}$ which is $\{$A,B,C,D$\}$. The total time cost is cost(swap(c1,b1) + swap(d1,a1)) + 1, since every swap needs three CNOT operations [13], it is totally 7.

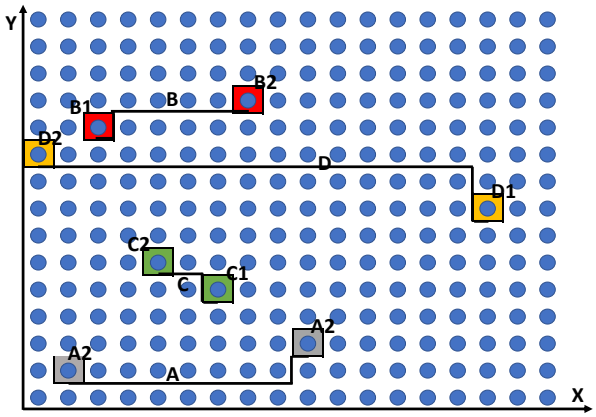

Fig. 2: optimization by swap

2) Change the braiding route

Based on Fig. 1, if we consider changing the route as shown in Fig. 3, then no two operations have any crossing so there is only one execution set $e_{1,1}$ which is $\{$A,B,C,D$\}$. And remember that changing the route length does not affect the execution time. In this case, the total cost is 1. In general, we need to consider the impact of changing the route of

one braiding carefully as it may cause more crossings with other fuure operations. So longer route might lead to more crossings.

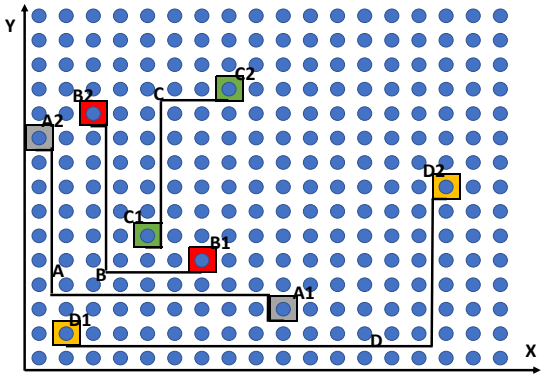

Fig. 3: optimization by changing braiding route

3) No Optimization

We did nothing for optimization, but we can save time for remapping, the execution set is {A},{B},{C}, and {D} sequentially, and the execution time is 4.

4) Optimization Trade-off

In summary, we want to find the minimum execution time based on different ways to divide S into s_i and e_{ij} sets. Route change may lead to more changes of execution sets in the future which might increase the total execution time. And based on Li's work in [14] swaps may have extra overhead and also lead to more ready sets in the future which might increase the execution time. So how to find a global optimal solution is a major problem here. We propose the following approaches.

- Backtracking: we can find all the possible ways to spit the set S, evaluate the cost of each, and obtain the minimum cost one.
- Heuristic greedy approach: [7], we can rank the braidings and set the priority for longest braiding path first or the one that intersects most with others first. We schedule the braidings that have no dependence and highest priority first, and schedule as many as we can in one cycle. Then we can split the braidings into different e_{ij} sets.

REFERENCES

[1] D. Knuth, "The art of computer programming, volume 2: Seminumerical algorithms, third edition," vol. 41, no. 2, pp. 379–417, 1997.

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[3] A. G. F. M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," in *PHYSICAL REVIEW A covering atomic, molecular, and optical physics and quantum information*, 2012.

[4] D. P. DiVincenzo, "Fault-tolerant architectures for superconducting qubits," *Physica Scripta*, vol. 2009, no. T137, p. 014020, 2009.

[5] R. Raussendorf and J. Harrington, "Fault-tolerant quantum computation with high threshold in two dimensions," *Physical review letters*, vol. 98, no. 19, p. 190504, 2007.

[6] S. B. Bravyi and A. Y. Kitaev, "Quantum codes on a lattice with boundary," *arXiv preprint quant-ph/9811052*, 1998.

[7] A. JavadiAbhari, P. Gokhale, A. Holmes, D. Franklin, K. R. Brown, M. Martonosi, and F. T. Chong, "Optimized surface code communication in superconducting quantum computers," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2017, Cambridge, MA, USA, October 14-18n*, 2017.

[8] J. Heckey, S. Patil, A. JavadiAbhari, A. Holmes, D. Kudrow, K. R. Brown, D. Franklin, F. T. Chong, and M. Martonosi, "Compiler management of communication and parallelism for quantum computation," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1. ACM, 2015, pp. 445–456.

[9] S. J. Devitt, A. M. Stephens, W. J. Munro, and K. Nemoto, "Requirements for fault-tolerant factoring on an atom-optics quantum computer," *Nature communications*, vol. 4, p. 2524, 2013.

[10] A. Javadi-Abhari, P. Gokhale, A. Holmes, D. Franklin, K. R. Brown, M. Martonosi, and F. T. Chong, "Optimized surface code communication in superconducting quantum computers," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 692–705.

[11] A. G. Fowler and S. J. Devitt, "A bridge to lower overhead quantum computation," *arXiv preprint arXiv:1209.0510*, 2012.

[12] Y. Shi, N. Leung, P. Gokhale, Z. Rossi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Optimized compilation of aggregated instructions for realistic quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: ACM, 2019, pp. 1031–1044. [Online]. Available: http://doi.acm.org/10.1145/3297858.3304018

[13] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. ACM, 2018, pp. 113–125.

[14] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 1001–1014.