Optimization of Quantum Circuit based on Surface Code

Group Members: Haojiang Zou, Junyin Zhang, Zhengtao Liu, Jingze Zhuang
Jingze Zhuang

March 14, 2021

Contents

Abstract

Surface code is widely believed to be a good choice for quantum circuit because of its scalability and error correction performance. Although considerable research has been devoted to the optimization of surface code, rather less attention has been paid to optimization by simply transforming the circuit to its equivalent . In this paper, we proposed a scheme to optimize error rate of surface-code-based quantum circuit. Then we introduce our program called "(Optimizer)" to actually perform the quantum circuit optimization.

1 Structure of Our Work

We first construct a surface code simulator to get the logical error rate from arbitrary surface code scheme and error model. Then combining a quantum circuit transformer that for arbitrary quantum circuit, gives its equivalent circuit, we may get an equivalent form that outperforms the original circuit when running on surface code. We seperate the simulator and the transformer by an interface in the form of Open QASM.

This paper is formed in two parts. Here the first part is about the surface code simulator. The second part is for the quantum circuit transformer.

In Section 2 and Section 3, we first give some background of surface code (mostly some references). Next in Section 4 and Section 5 we will introduce our simulator.

2 Basics of Surface Code

Surface Code was first proposed by Kitaev, in its original form of toric code. A fine paper describing the surface code would be [2]. Other references is listed in [4, 5, 1].

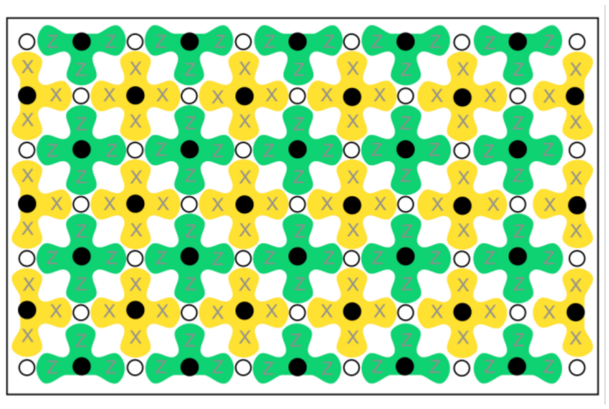
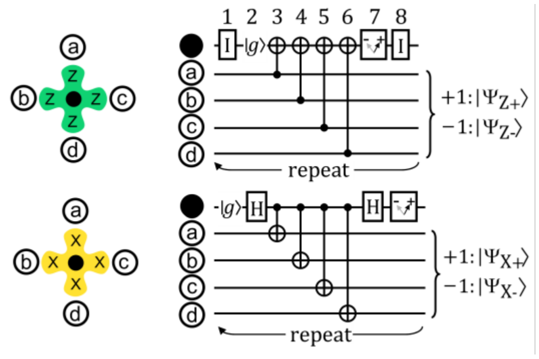


Figure 1: Geometry of surface code



3 Surface Code Background

Most simulation of surface code is focused on its perfomance of information storing (See search result of "surface code simulation" in google search and github). The reason may be its relationship with condensed matter theory and its topological significance.

However, few attention has been focused on simulating the performance of computing based on surface code. What's more, notable works [3, 6] did not optimize quantum circuit in our way.

4 Surface Code Simulator: Code Explanation

- In /src/geometry.jl, we define the geometry of our code as Fig. 1 shows.
- In /src/model.jl, we define the surface code model and the state ("Where the error occurs" and "Where the error is detected")
- In /src/utils.jl, we constructed physical qubits' error generator, error detection simulator, error correction simulator, and correction result judger to see if error correction successes. In the error correction simulator, we used Edmond's Minimum Weight Perfect Matching algorithm that highly speeds up our program.
- In /src/controller.jl, we constructed automatic error rate calculator that can calculate error rate to arbitrary given precision. Thus, we get error threshold that is important in physicists' scope.
- In /src/braiding.jl, using the surface code for computation that is defined in the above, we constructed two qubits gate simulator that is able to perform braiding operation (CNOT gate) between two qubits.

5 Surface Code Simulator: Results

In Fig. 2, we show our result of error threshold. This fits well with the prediction by condensed matther theory in physics.

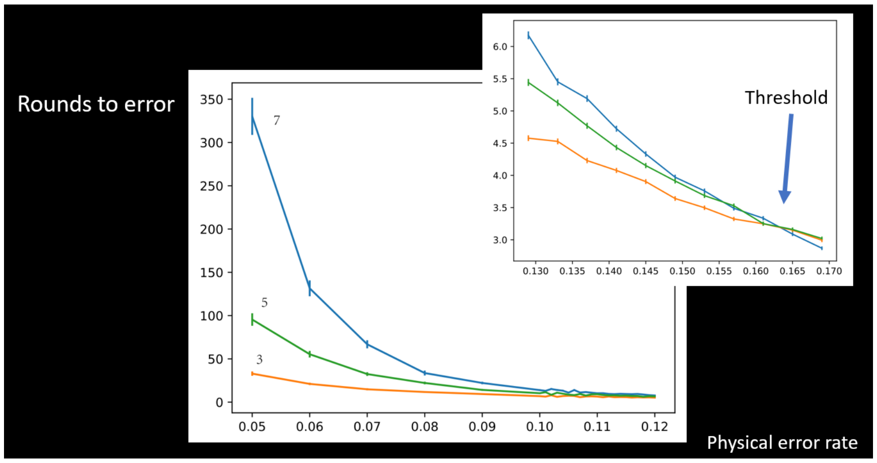


Figure 2: Error threshold of a surface code model (for storage)

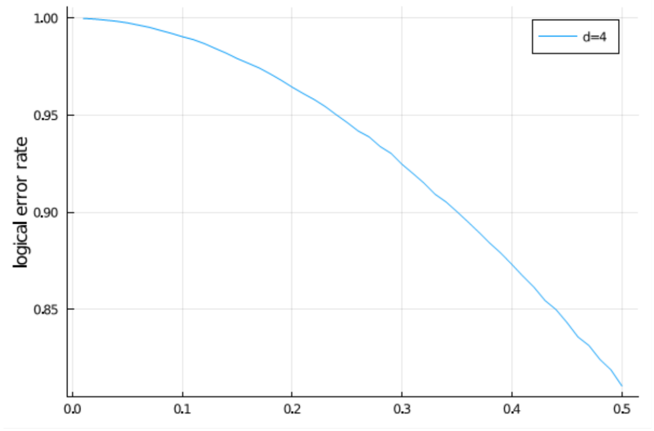


Figure 3: Error rate of a surface code model (for computation)

In Fig. 3, we show our result of error rate of CNOT gates in surface code.

6 Future Aspects

There is still much work to do to achieve our goal. First, we must say that our code has not finished yet. Firstly, up to only two logical qubits can be simulated in this version. We may need to support more qubits in the future. Luckily, the computation speed of our algorithm will helped us a lot in expansion of the scale of our simulator. Second, more quantum operation should be added. The available set of operations is limited in surface code model. Third, the error model may need refinement. For example, the error rate would be more reliable if we refer to the near-term real quantum platform.

References

[1] Benjamin J Brown. A fault-tolerant non-clifford gate for the surface code in two dimensions. *Science advances*, 6(21):eaay4929, 2020.

[2] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[3] Casey Duckering, Jonathan M Baker, David I Schuster, and Frederic T Chong. Virtualized logical qubits: A 2.5 d architecture for error-corrected quantum computing. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 173–185. IEEE, 2020.

[4] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[5] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[6] Alexandru Paler. Surfbraid: A concept tool for preparing and resource estimating quantum circuits protected by the surface code. *arXiv preprint arXiv:1902.02417*, 2019.

量子线路子图替换

目的

通过子图变换可以生成具有等效功能的不同电路，从而对之前的 `surface code` 算法结果进行校验。

替换的另一目的为提供一种可行的物理捷径：通过对等效电路替换子图的研究，可以尝试找到物理线路误差最小，性能最优的线路。

设计输入输出

程序使用 Python 完成。输入与输出都是 OpenQASM 格式。

我们的最初设想应该是输入一个格式的量子线路，以获得更多的量子线路。但是表示量子线路的方式有很多，而同时实现量子程序的语言也很多（仅以我们的项目为例，就至少用到了 Python，C++，Julia 等）。为了提高兼容性，我们需要选择一种较为通用的格式来进行语言中的转换（在我们的项目中，主要是通过 Python 脚本生成 Julia 可用的量子线路格式作为输入）。而 OpenQASM 是一个比较成熟的量子汇编语言，可以方便的在各种语言中读取。

算法简述

程序逐行读入 OpenQASM 程序，每行代表一个量子门电路。之后程序遍历所有电路，并尝试与我们的规则进行匹配，并标记出所有可以匹配的点。

遍历完成后即可获取所有能替换子图组成，从而随机选取进行替换。重复以上操作，即可得到不同的子图。对我们的算法来说，主要难点在于如何定义我们的规则，以及如何找到可替换的子图。

定义规则

下为我们定义的一个量子变换规则：

```
FROM
h 1 2
cx 2 1 2
tdg 1 2
cx 2 0 2
t 1 2
cx 2 1 2
tdg 1 2
cx 2 0 2
t 1 1
t 1 2
h 1 2
cx 2 0 1
t 1 0
tdg 1 1
cx 2 0 1
```

```
TO
ccx 3 0 1 2
```

规则应该为一个子图转化为一个子图，为此，我们将要变换子图放在 `FROM` 中，将变换结果放入 `TO` 中。事实上，这个过程是可逆的。

受 OpenQASM 启发，我们的规则表示门电路的方式与 OpenQASM 类似。每一行第一列用与 OpenQASM 相同的符号表示门电路种类，并在第二列声明该门电路用到量子比特数量。第三列及以后分别表示量子比特编号与对应参数（如角度）。

定义好规则文件后，利用其与 OpenQASM 的相似性，我们可以方便的与原电路结合起来，执行替换，这样就实现了 rule 到抽象规则的转化。

重新映射规则

仅仅有规则是不够的，首先在电路中量子比特位置与规则不一定相同，其次对于那些可以忽略的门电路，我们的算法应予以排除，从而找到尽可能多的等效电路。为此我们通过将规则与其映射的方式实现了较为全面的查找。

当尝试用规则 1 的逻辑门 `cx 2 q0 q1` 来匹配实际逻辑门 `cx 2 a0 a1` 时，将 `a0` 映射为 `q0`，`a1` 映射为 `q1`。这样与下一个逻辑门 `cx 2 q0 q2` 来匹配实际逻辑门 `cx 2 a3 a0` 时，`a0` 仍将对应原电路的 `q0` 比特，`a3` 则为 `q3` 比特。记忆上述变换进行查找，直到匹配到不符电路或匹配成功。

匹配完成后，程序移动到下一个门电路，进行下一轮匹配。

跳过无关门电路

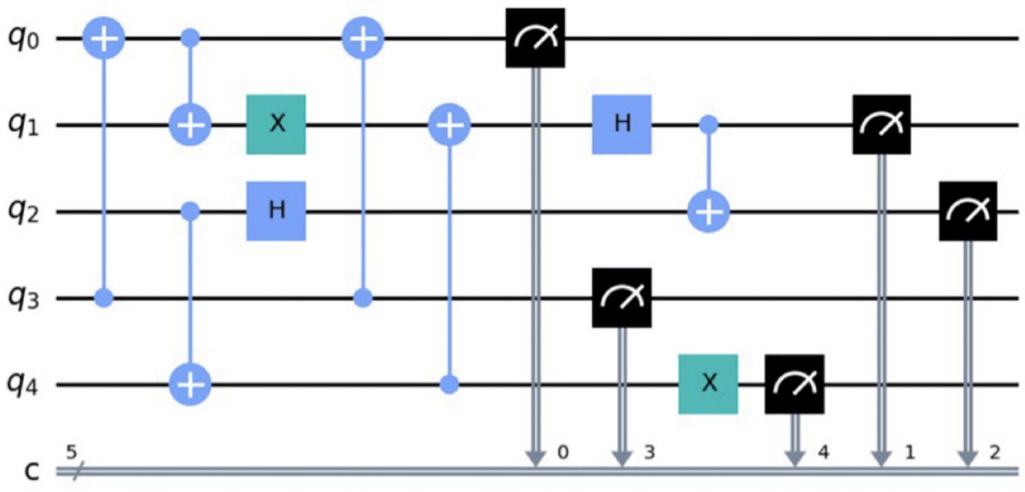
假设门电路第一位匹配成功，用到的量子比特计为 `q1` `q2`，但是下一个逻辑门电路用到的比特为 `q3`，而之后门线路中不再用到 `q3` 比特，那么我们可以视为此门电路没有参与此次匹配，按照规则，我们可以 “去除“ 这个门电路以进行下一步匹配。

如果匹配成功，则我们可以将无关量子逻辑门电路提到最前或者最后，而不影响电路功能。

通过上述两步，我们可以较为全面的检测匹配到的门电路。

算法实例举例

以下面线路为例：

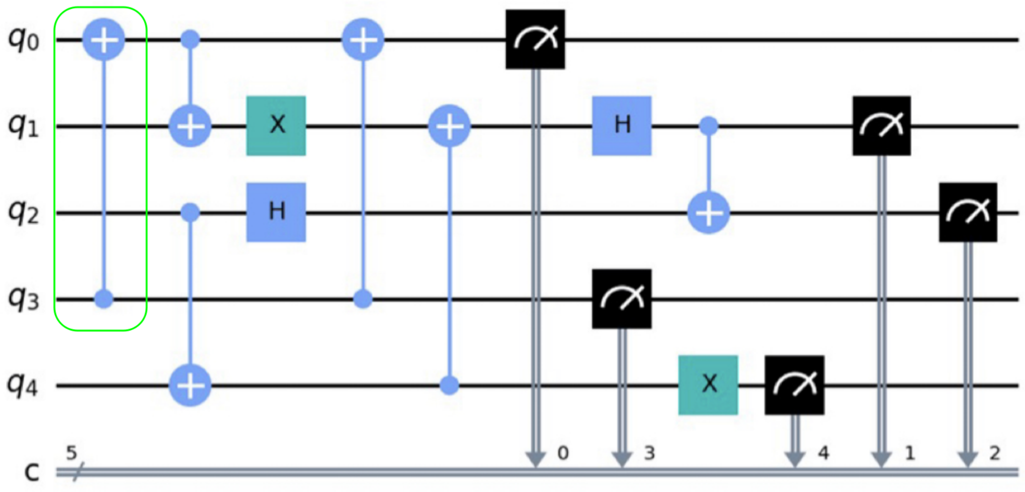


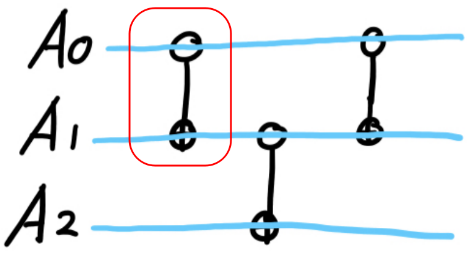
按照我们在上一个流程定义的规则，其可找到的一个等效规则为：

```
FROM
cx 2 0 1
cx 2 0 2

TO
cx 2 1 2
cx 2 0 1
cx 2 1 2
```

首先匹配第一个门电路：

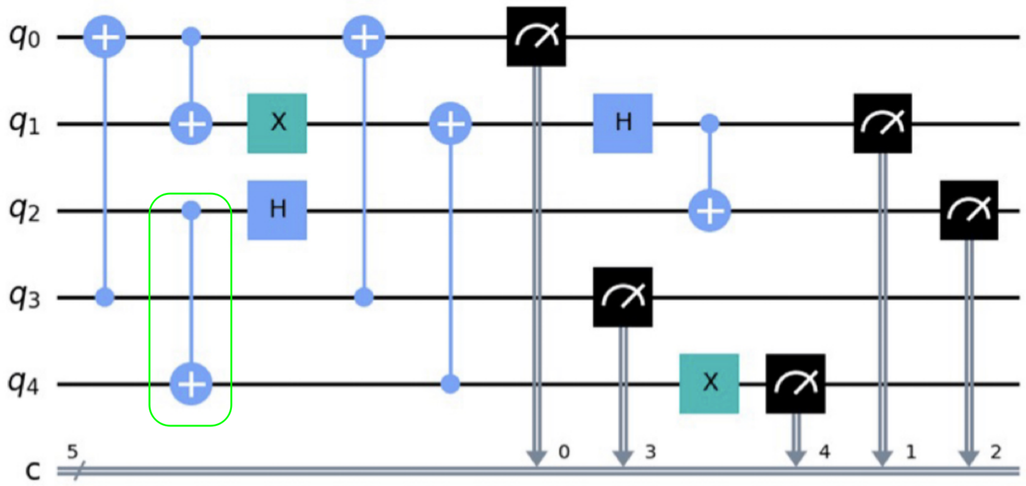


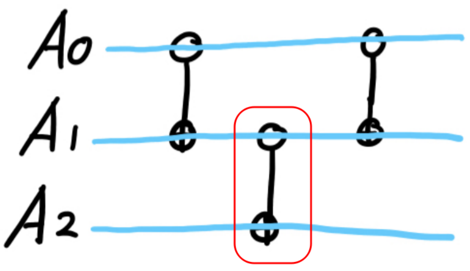


按照我们先前的映射规则，我们可以看到有：

```
A0 = q3
A1 = q0
```

下一步匹配：



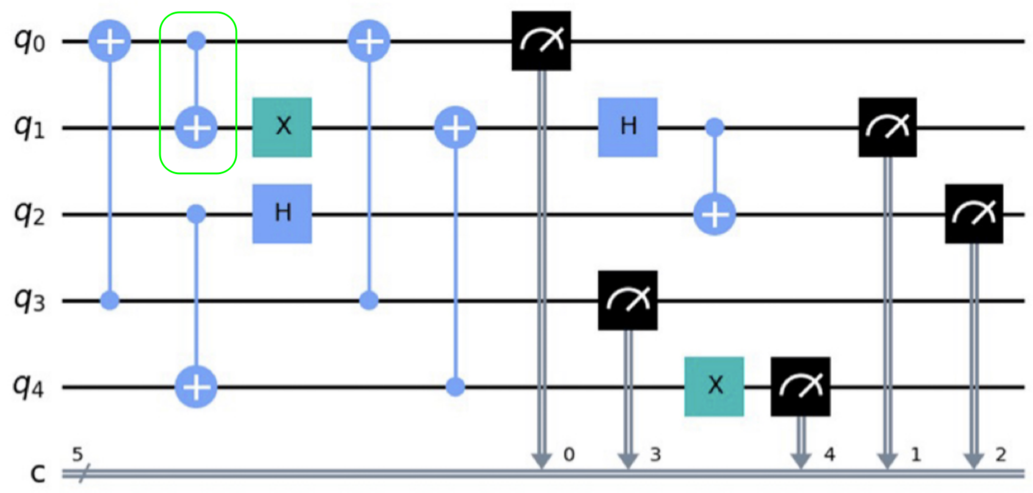


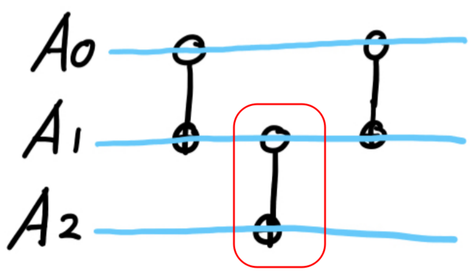
这一步得到的映射为：

```
A0 = q3
A1 = q0

A1 = q2
A2 = q4
```

显然，`A1 = q0` 规则与 `A1 = q2` 是矛盾的，此门电路为无关，忽略并查找下一个门。

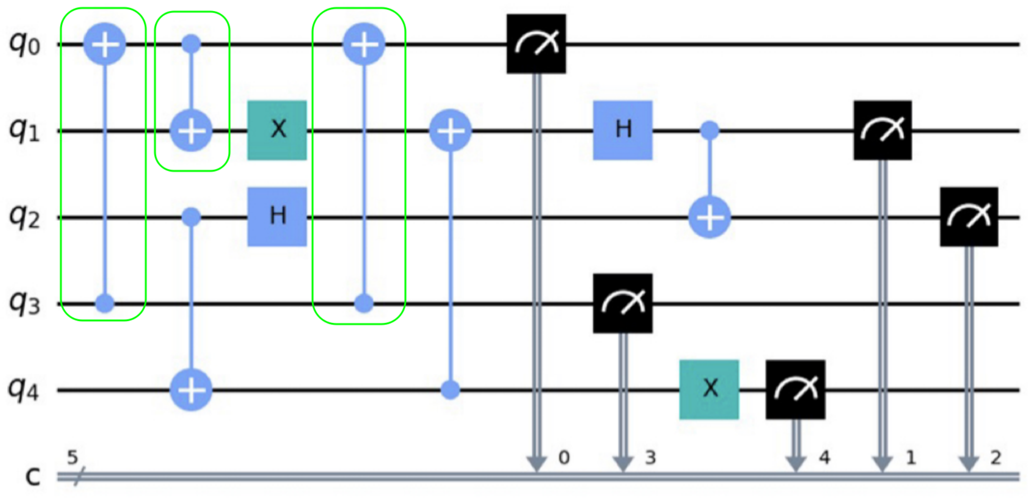


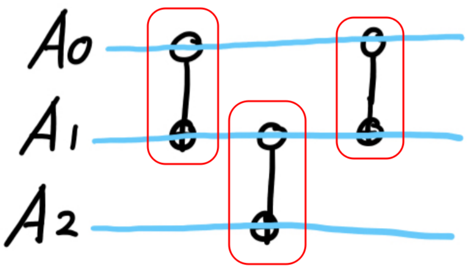


得到不冲突规则：

```
A0 = q3
A1 = q0
A2 = q1
```

最后可以得到可替换的门电路为：





其对应的变换规则为：

```
A0 = q3
A1 = q0
A2 = q1
```

可进一步进行的优化

架构优化

我们可以使用现成的 qPanda 包或者 Qiskit 包来进行量子线路的输入与输出处理。从而得到更稳定的结构。这也是我们的最初设想，比较遗憾的是，我们发现将自定义门电路符号与 Qiskit 门电路对应起来仍需要复杂的逻辑，甚至更复杂。为此我们经过权衡考虑，最后决定直接读写 OpenQASM 文件。但是这容易导致的一个问题是当 OpenQASM 格式发生改变等情况可能无法良好迭代。

算法改进

事实上我们注意到变换可以分为两类，一类为单个门电路子图变换，一类为多个比特门的子图变换。对于单门变换，只要有足够的量子比特，可以直接变换。对于多个门电路，则需要进行匹配操作。

为此可以进行如下改进：首先找到所有可行的多门子图，在完成替换后，随机选择可替换的单门子图，进行替换。这样可以快速生成等效门电路。

此外，由于用到了匹配规则，可以考虑使用正则算法对匹配方式进行优化，从而更有效准确的进行匹配。

总结

通过构建程序包 `simplify_pack` ，我们成功的实现了对 OpenQASM 格式输入的量子线路按照给定规则查找子图，并以相同格式输出算法。

同时算法具有一定的可拓展性，使用者可以通过编写自定义规则来实现更多种类的替换规则。