Ecmas: Efficient Circuit Mapping and Scheduling for Surface Code

Mingzheng Zhu[†], Hao Fu[†], Jun Wu[†], Chi Zhang[†], Wei Xie[†], Xiang-Yang Li[†‡]
[†] University of Science and Technology of China, China
[‡] Hefei National Laboratory, University of Science and Technology of China, Hefei 230088, China

***Abstract*—As the leading candidate of quantum error correction codes, surface code suffers from significant overhead, such as execution time. Reducing the circuit's execution time not only enhances its execution efficiency but also improves fidelity. However, finding the shortest execution time is NP-hard.**

In this work, we study the surface code mapping and scheduling problem. To reduce the execution time of a quantum circuit, we first introduce two novel metrics: Circuit Parallelism Degree and Chip Communication Capacity to quantitatively characterize quantum circuits and chips. Then, we propose a resource-adaptive mapping and scheduling method, named `Ecmas`, with customized initialization of chip resources for each circuit. `Ecmas` can dramatically reduce the execution time in both double defect and lattice surgery models. Furthermore, we provide an additional version `Ecmas`-ReSu for sufficient qubits, which is performance-guaranteed and more efficient. Extensive numerical tests on practical datasets show that `Ecmas` outperforms the state-of-the-art methods by reducing the execution time by 51.5% on average for double defect model. `Ecmas` can reach the optimal result in most benchmarks, reducing the execution time by up to 13.9% for lattice surgery model.

***Index Terms*—Surface Code, Compilation, Execution Time**

I. Introduction

Quantum algorithms offer exponential speedup over classical algorithms in various fields such as machine learning [14], [18], [30], simulation [11] and cryptography [31]. One of the obstacles to achieving such advantages is the inevitable errors of quantum hardware. The error rate of the state-of-the-art superconducting quantum devices is around 10^{-3} per operation [2], [13], [37], which falls far short of meeting the demands of practical applications [12]. One approach to handle these errors is quantum error correction (QEC), which establishes the fault-tolerant computational framework [32]. Surface code [5], [7], [22] currently stands as the most promising QEC code, highlighting a threshold error rate of up to 10^{-2}. Its natural 2-D nearest-neighbor structure makes it well-suited for implementation on superconducting chips.

Applying surface code to protect a quantum circuit involves converting the circuit into an encoded form. Unlike the circuit transformation typical in the NISQ (Noisy Intermediate-Scale Quantum) era, surface code transformation necessitates mapping a single logical qubit to a cluster of physical qubits, known as tiles [19]. As a result, the conditions for executing logical operations differ significantly. For instance, a CNOT gate no longer requires physical qubits to be adjacent on the chip. Instead, it can be implemented by establishing a non-intersecting path between two distinct tiles, called qubit communication. These requirements call for developing an efficient and specialized transformer to transform a quantum circuit into a surface-code-encoded circuit.

The transformation process has two stages: initialization and scheduling. In the initialization stage, the transformer needs to allocate tiles for each logical qubit and allocate channels for communication. In the scheduling stage, the transformer determines the specific execution schemes for each operation. Most operations can be performed within the tiles [10], except T gate and CNOT gate, which are the most resource-consuming logical operations. The substantial overhead of T gates stems from their inability to be fault-tolerantly executed, thereby necessitating the use of supplementary *magic state distillation* circuits [4]. Through extensive research efforts [8], [26], this overhead has been considerably reduced. However, the time delay induced by CNOT gates is severe, particularly in circuits where quite a lot of CNOT gates can be executed in parallel, such as Ising circuits [20], [29] and the QDNN circuits [34]. This significantly influences the fidelity of the execution result of the surface code circuit. With the same physical error rates and the code distance, a shorter execution time yields higher result fidelity. Thus, an essential goal of transformation is to reduce the execution time of the circuit. However, finding the shortest execution time of a circuit is NP-hard (as proved in Theorem 1), which makes it a non-trivial task for finding an effective result efficiently.

Executing CNOT gates can be simplified as constructing a path between the two involved tiles, regardless of the specific encoding scheme, i.e., double defect model [10] or lattice surgery model [3]. Logical qubits are represented as small boxes in Fig. 1, and channels are the residual regions used to establish the paths (depicted by the lines). CNOT gates can be completed within one clock cycle, regardless of the path length. Simultaneous execution of CNOT gates requires non-intersect corresponding paths.

Many works focus on the paths of CNOT gates to reduce the latency caused by path conflicts [3], [17], [19]. Braidflash [19] reduces the latency of CNOT gates on the critical path by assigning priority to CNOT gates to reduce the delay of the conflicts. AutoBraid [17] identifies specific patterns to find non-intersecting paths and EDPCI [3] draws inspiration from the concept of edge-disjoint paths. However, they have all overlooked a crucial aspect: within the context of surface code, the communication resources on the chip are software-defined. We use **bandwidth** to represent the width of the channel, with which we can adaptively adjust communication resources for varying circuits.

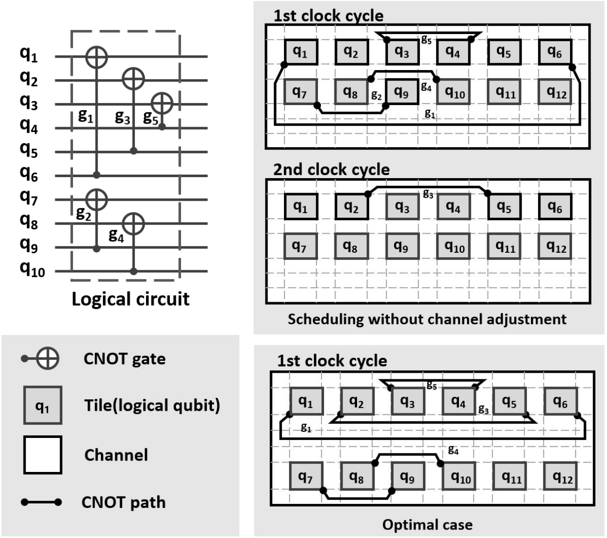


Figure 1: Motivation example: a logical quantum circuit and its corresponding surface code encoded circuit.

Motivation Example: As shown in Fig. 1, five independent gates are ready to be executed. Each gate's path requires a certain width of physical qubits during execution. Due to the lack of quantitative analysis on the channels, the path occupies the entire channel in the scheduling process of previous works. As a result, no five disjoint paths allow these gates to be executed simultaneously. However, in the optimal case, one clock cycle is enough to execute the circuit with the same chip and tile placement by better allocating the channel resource.

In this paper, we study the surface code circuit mapping and scheduling problem. We propose resource-adaptive transforming methods `Ecmas` which reduce the execution time of circuits on target chips by customizing channel resources for each circuit. The main idea is to characterize the circuit and the chip and then schedule communication resources based on the circuit's requirements. Our contributions are summarized as follows:

- We formulate the surface code circuit mapping and scheduling problem and analyze the computational complexity of double defect model.
- We define *Circuit Parallelism Degree* and *Chip Communication Capacity* to quantitatively characterize circuits and chips. Further, we introduce *bandwidth* to customizing channel resources for each quantum circuit.
- We propose resource-adaptive mapping and scheduling methods that can be applied to both double defect and lattice surgery models. With sufficient physical qubits, `Ecmas` offers `Ecmas`-ReSu which can provide performance-guaranteed results efficiently.
- We evaluate `Ecmas` for circuits from IBM Qiskit [29], QASMbench [24], etc. With the same chip resource configuration, `Ecmas` eliminates 51.5% of the execution time on average and 67.3% at most compared with Autobraid [17] for double defect model. `Ecmas` could find the optimal result for most benchmarks for lattice surgery model. Compared with EDPCI [3], `Ecmas` can achieve optimizations up to 13.9%.

The rest of this paper is organized as follows. We introduce the background in Section II and then formulate the surface code mapping and scheduling problem in Section III. We describe our methods in Section IV and evaluate their performance in Section V. Related works and conclusion are given in Section VI and Section VII.

II. Background

In this section, we present a brief overview of quantum error correction and surface code model.

A. Quantum Error Correction

Quantum programs can be described by the quantum circuit model, which consists of a sequence of quantum gates performed upon a collection of qubits. Qubits are the fundamental units in quantum computing which can be represented by a normalized vector. Quantum gates are unitary operations that operate on qubits.

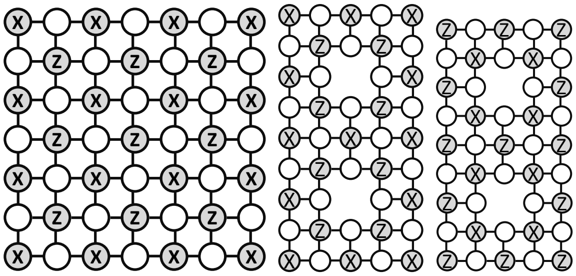
However, quantum computing suffers from the inevitable noise of interactions with the environment and imprecise operations. QEC codes are necessary to build fault-tolerant quantum computing. It encodes a logical qubit with multiple physical qubits, improving reliability. The noise of the quantum system appears not only in the communication process but also in the computation process. Therefore, the quantum circuit must run under the protection of QEC. QEC codes should detect and correct errors periodically during the execution.

B. Surface Code

Among various QEC codes, surface code is a prominent candidate for achieving fault-tolerant quantum computation in superconducting implementations. It has a high threshold of around 1% and alignment with 2D architectures, making it a feasible error correction code for practical demonstrations on real machines [1], [23], [39].

As shown in Fig. 2, surface code is realized on a 2D lattice of physical qubits, including data and measurement qubits. Data qubits store quantum states, while measurement qubits identify error occurrences. Based on the measurement circuit, measurement qubits are categorized as X-stabilizers and Z-stabilizers. During the execution, measurement circuits are periodically executed to detect the errors. The time for executing one measurement circuit is called a surface code cycle. Surface code can be classified as double defect [10] and lattice surgery [3] based on the different approaches to creating logical qubits.

1) Double Defect Model: In double defect model, a logical qubit is created by turning off two defects of the same type. According to the type of defects, the logical qubit is initialized into X-cut or Z-cut, as shown in Fig. 2b and Fig. 2c. Code distance d determines the number of errors that surface code

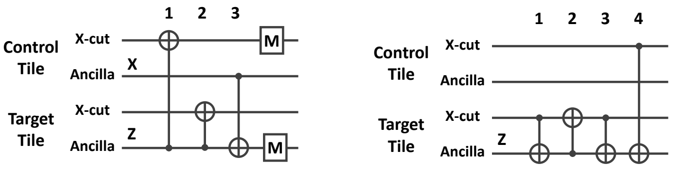


(a) (b) (c)

Figure 2: (a) Surface code implementation on 2-D lattice, the white circles are data qubits and the gray circles are measurement qubits, (b) X-cut tile, (c) Z-cut tile.

can detect and correct. All single-qubit gates can be executed in software or locally, only involving physical qubits around its two defects under the assumption in [19] that a steady supply of magic state qubits is at the location of the data. We denoted these physical qubits as tiles and the rest of the qubits on the chip as channels.

CNOT gate requires communication between the control and target qubit, achieved by performing braiding operations in the channels. A braiding operation turns off the involved measurement qubits on the braiding path. It follows the topological rules: the braiding paths are equivalent as long as the starting and ending tiles are the same. Braiding operations of any length can be executed within $2d$ surface code cycles, equivalent to one clock cycle. The braiding operation can only be performed between logical qubits with different cut types. In practice, each tile contains two double-defect logical qubits, one for computation and one for ancilla. There are two ways to perform a CNOT gate with qubits of the same cut type. One is to use three braiding operations with ancilla qubit, as shown in Fig. 3a. The other is to modify the cut type of the tile and then perform the braiding operation, as shown in Fig. 3b. They require three cycles and four cycles, respectively.



(a) Direct implementation (b) Modifying implementation

Figure 3: CNOT gate between logical qubits of same cut type: (a) three braiding operations without cut type changing, (b) changing the cut type and executing the CNOT gate.

2) Lattice Surgery Model: Lattice surgery eliminates the holes within tiles and uses the rotated surface code (as shown in Fig. 5b) to reduce the requirement of qubit resources for surface code with the same code distance. CNOT gates in lattice surgery are attained by conducting ZZ measurements between neighboring tiles. A straightforward approach for a CNOT gate at a distance k involves continuously swapping logical qubits until they are adjacent, requiring a minimum of $k \times d$ surface code cycles to complete. Another method involves constructing Bell states using ancilla qubits for execution, achievable within $2 \times d$ surface code cycles i.e. one cycle as shown in Fig. 4.

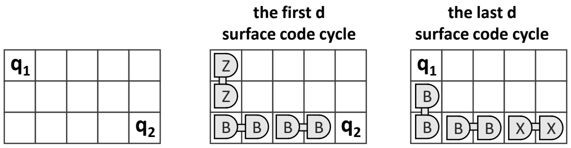
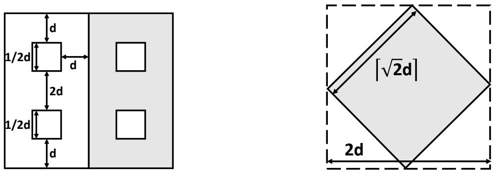


Figure 4: The CNOT gate implementation in lattice surgery model by constructing Bell states.



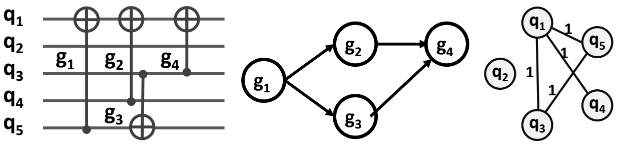
(a) (b)

Figure 5: Simplified tile models: (a) double defect model, (b) lattice surgery model.

III. System Model and Problem Formulation

In this section, we formally define the surface code mapping and scheduling problem for both double defect and lattice surgery models and demonstrate the complexity of the problem under double defect model.

Quantum circuit: We consider an input quantum circuit P with n logical qubits (Fig. 6a). Since single-qubit gates can be implemented by software or locally in tile, we only consider CNOT gates in this work. Generally, P can be represented as a directed acyclic graph (DAG) G_P, as shown in Fig. 6b. In G_P, each node is a CNOT gate, and edges indicate the dependency between gates. The critical path length of G_P is the circuit depth, denoted as α. The communication graph G_C is another representation of a quantum circuit, as shown in Fig. 6c, where each vertex is a logical qubit, and edges indicate CNOT gates between the qubits, and the weight of the edge is the number of the corresponding CNOT gates.



(a) (b) (c)

Figure 6: Three representations of quantum circuit: (a) original, (b) DAG, (c) communication graph.

Quantum chip: We assume that the topology of a quantum chip is the 2D lattice of the physical qubit, where each qubit is associated with four adjacent qubits, except the qubit on the chip's boundary. We use $L_{m_1 \times m_2}$ to denote the 2D chip with m_1 rows and m_2 columns of physical qubits.

Surface Code Encoded Circuit: The encoded circuits P^S should satisfy the following two constraints. First, the execution scheme should be equivalent to the logical circuit, i.e., all gates are scheduled, and the scheduling order is consistent with the topological sort of gates in G_P. Second, the CNOT paths of the gates executed in one cycle do not intersect. The execution time of a circuit is $\Delta \times 2d \times \tau$, where Δ is the cycle number and τ is the execution time of each surface code cycle. Since d and τ have the same effect on different mapping and scheduling methods, we simplify the execution time as cycle number Δ.

Surface Code mapping and scheduling Problem: Given an input quantum circuit P, a specific quantum chip $L_{m_1 \times m_2}$ and the required code distance d find an initial mapping and the execution scheme that satisfy the surface code circuit constraints with the cycle number of circuit Δ_{P^S} be minimized.

Next, we will refine the models for double defect and lattice surgery approaches. We further provide detailed descriptions of each model and offer formal problem definitions.

A. Double Defect Model

Tile: A tile is a square array of $5d \times 5d$ physical qubits, as shown in Fig. 5a, each tile contains two logical qubits, one for mapping logical qubits and one for *ancilla*. We use $(T_{a,b}, Cut_i)$ to denote tile T_i, where (a, b) is the position of the upper left corner of the tile and Cut_i is its cut type.

Channel: Channel is used to perform braiding operations. Each braiding path requires a width of $2.5d$ physical qubits. We consider the occupation of a braiding path within a channel as a lane. We introduce **bandwidth** to characterize the number of lanes in each channel. The bandwidth of a channel C_i is $\lfloor \frac{W_i}{2.5d} \rfloor$, where W_i is the number of physical qubits in the width of the channel C_i. Then, we consider the minimum bandwidth of channels within the chip as the chip's bandwidth.

Fig. 7 illustrates the process of surface code transformation for the double defect model. We present the formal description as follows.

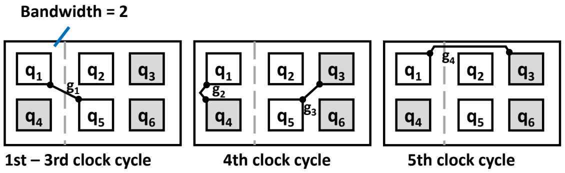


Figure 7: A five-step execution scheme for the quantum circuit in Fig.6a using the double defect model, where the gray boxes are for X-cut tiles and the white boxes are for Z-cut tiles.

Problem 1: **Initialization Problem for Double Defect.**
Input: An input logical circuit P, a 2D lattice chip $L_{m_1 \times m_2}$, the required code distance d and a natural number k.
Output: Whether there is an initial tile mapping $T_{mapping} = \{q_i \rightarrow (T_{a,b}, Cut_i)\}$ such that the number of cycles of the optimal surface code encoded circuit P^S_{OPT} is upper bounded by $\alpha + k$, namely, $\Delta^{P^S_{OPT}} < \alpha + k$.

Problem 2: **Scheduling Problem for Double Defect.**
Input: An input logical circuit P, a 2D lattice chip $L_{m1 \times m2}$ and an initial tile mapping $T_{mapping}$.
Output: A surface code encoded circuit P^S with its number of cycles Δ^{P^S} minimized.

Hardness:

Theorem 1: The surface code tile initialization problem for double defect model is NP-hard.

Proof sketch:

We reduce the Initialization Problem for double defect model into a 3-SAT problem. For more details, please refer to Appendix A.

B. Lattice Surgery Model

Tile: As shown in Fig. 5b, each small box represents a tile that can be mapped as a logical qubit, with $\lceil \sqrt{2}d \rceil \times \lceil \sqrt{2}d \rceil$ physical qubits. We denote tile T_i by $T_{a,b}$, where (a, b) represents the upper left position of this tile.

Channel: Each channel is composed of tiles, which are ancilla logical qubits to generate Bell states for communication. Since both logical qubits and channels are constructed from tiles, the width of a path and a tile are the same, consisting of d physical qubits. The bandwidth of the channels C_i is given by $\lfloor \frac{W_i}{\lceil \sqrt{2}d \rceil} \rfloor$. The chip's bandwidth is the minimal bandwidth of its channels.

Fig. 8 shows the process of surface code transformation for the lattice surgery model. Below, we present the formal depiction of these problems.

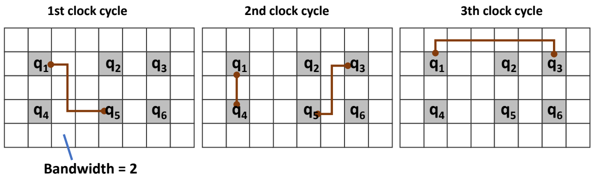


Figure 8: A three-step execution scheme for the quantum circuit in Fig. 6a using the lattice surgery model.

Problem 3: **Initialization Problem for Lattice Surgery**
Input: An input logical circuit P, a 2D lattice chip $L_{m_1 \times m_2}$, the required code distance d and a natural number k.
Output: Whether there is an initial tile mapping $T_{mapping} = \{q_i \rightarrow T_{a,b}\}$ such that $\Delta^{P^S_{OPT}} < \alpha + k$, namely, the number of cycles of the optimal surface code encoded circuit P^S_{OPT} is upper bounded by $\alpha + k$.

Problem 4: **Scheduling Problem for Lattice Surgery.**
Input: An input logical circuit P, a 2D lattice chip $L_{m_1 \times m_2}$ and an initial tile mapping $T_{mapping}$.
Output: A surface code encoded circuit P^S with its number of cycles Δ^{P^S} minimized.

Hardness: Herr *et al.* [15] demonstrated that the complexity of surface code mapping and transforming problem is NP-complete for lattice surgery model.

IV. System Design

It is a non-trivial task to optimize circuit mapping and scheduling on limited qubit resources. To address the problem, firstly, we introduce two novel metrics: *Circuit Parallelism Degree* and *Chip Communication Capacity* (Section. IV-A) to characterize quantum circuits and chips. Then, we propose resource-adaptive algorithms `Ecmas` (Section. IV-B) with customized initialization of chip resources for each circuit. Further, with sufficient physical qubits on the chip, `Ecmas-ReSu` can have a shorter transforming time and performance-guaranteed result. An overview of our comprehensive toolflow is shown in Fig. 9.

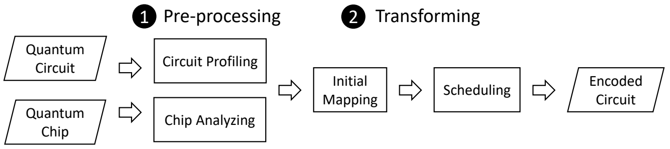


Figure 9: Overview of `Ecmas`.

A. Pre-processing

1) Quantum Circuit Profiling: Different quantum circuits may have various demands on communication resources. We introduce *Circuit Parallelism Degree* (denoted as $\mathbb{PM}$) to characterize the maximum demand of communication resources of a circuit.

Definition 1: Circuit Parallelism Degree: Given a quantum circuit P, $G_P = (V, E)$. A partition π is to divide nodes $v \in V$ into Δ_{G_P} disjoint set $V_1, V_2, ..., V_{\Delta_{G_P}}$, such that for $u \in V_i$ and $v \in V_j$, if $(u, v) \in E$, then $i > j$. $\mathbb{PM} = \min_\pi \max_{i=1}^{\Delta_{G_P}} |V_i|$

Finding $\mathbb{PM}$ is equivalent to given n tasks and their precedence constraints, minimizing the number of machines used while the whole schedule is of minimum length. Finke [9] has proved that this is NP-complete.

We propose a heuristic algorithm (Algorithm Para-Finding) to find the circuit's estimate Circuit Parallelism Degree $\widetilde{\mathbb{PM}}$ and the corresponding execution order. Our methods use layers to keep track of the execution order of gates, where $layer_1$ represents the operations to be performed in the first time cycle. For any gate i, we record two values, the highest and lowest layers, that the gate can be scheduled. Layers are determined by the gate's parents and children nodes, denoted as $parent_i$ and $child_i$. $Low_i = \max_{j \in parent_i} Low_j + 1$ and $High_i = \min_{j \in child_i} High_j - 1$. Then, we calculate the difference between the gates' high and low values and choose the gate with the smallest difference. For this gate, we schedule it to the layer with the fewest gates to execute in all the possible layers. After that, we update the low value of its child nodes and the high value of its parent node. We repeat this process until all the gates are scheduled. The maximum number of gates per layer is $\widetilde{\mathbb{PM}}$ of this circuit.

2) Quantum Chip Analyzing: We define the *Chip Communication Capacity* to characterize the number of parallel CNOT gates supported by a chip, denoted as $\mathbb{C}$. According to [17], any three CNOT gates can be executed simultaneously. As we refine the chip model, we generalize the previous theorem to the case that any $\lfloor \frac{b-1}{2} \rfloor + 3$ braiding operations can be executed simultaneously where b is the chip's bandwidth.

Definition 2: Chip Communication Capacity: Given a quantum chip, $\mathbb{C}$ is the max number u that for any u independent CNOT gates with an arbitrary placement of tiles, there exists a simultaneous path schedule for all CNOT gates.

Theorem 2: For a chip with bandwidth b, given an arbitrary placement of the operand qubits, there exists a simultaneous paths schedule for $\lfloor \frac{b-1}{2} \rfloor + 3$ gates.

Proof: According to Autobraid [17], any three CNOT operations must be able to execute simultaneously on a chip with bandwidth 1. The path of the additional CNOT gate has to intersect with others if and only if one involved tile is inside a ring and the other is outside. A ring is composed of paths and fully occupied channels. Increasing the bandwidth of each channel on the chip by two would break this ring and enable a path connecting two arbitrary tiles on the chip. Therefore, when the chip's bandwidth is b, paths exist for $\lfloor \frac{b-1}{2} \rfloor + 3$ CNOT operations to be executed in parallel.

B. Transforming

1) Initial Mapping: To generate a preferred tile location mapping, we employ the following three steps (Line1-3 in Algorithm1):

Shape Determining. First, we determine the shape of the logical tile array, i.e., whether to initialize it as a 3×3 array or a 2×4 array for a circuit with eight logical qubits when both schemes are available on the chip. Then, We select an array shape with the minimum perimeter. As shown in Fig. 10a, we choose the 3×3 tile array.

Mapping Establishing. Secondly, We map each logical qubit to its corresponding logical tile according to communication cost, as shown in Fig. 10b. The communication cost is calculated by cost function $f = \sum_{i,j} \gamma_{i,j} \times l_{i,j}$, where $l_{i,j}$ represents the Manhattan distance between the two tiles T_i and T_j, and $\gamma_{i,j}$ is the number of $CNOT_{i,j}$ in the circuit. Mapping qubits that frequently communicate together can effectively reduce the communication cost. Here, we employ the Metis [21] method, an iterative graph partitioner, to generate mappings based on the qubit communication graph G_C and tile array. Due to the stochastic steps in the mapping generation, we generate multiple mappings and select the one with minimal communication cost as our final result.

Bandwidth Adjusting. Finally, we assign the rest of the qubit resources to each channel based on their occupancy status, as illustrated in Fig. 10c. We pre-execute each gate in the circuit to record its shortest path without considering the non-intersecting restrictions. Then, we increase the bandwidth for channels that perform the most paths. In most cases, this process effectively reduces the wait caused by channel resource occupation.

2) Scheduling: Considering the qubit resources on the target chip may be limited or sufficient, we design two algorithms to maximize the utilization of resources.

Scheduling for limited Resources. When the resources of physical qubits are limited, i.e., when $\widetilde{\mathbb{PM}} > \lfloor \frac{b-1}{2} \rfloor + 3$, it may be difficult to find non-intersecting paths to execute all current executable gates. However, the number of children in gates of currently executable gates varies. We assign priorities to these

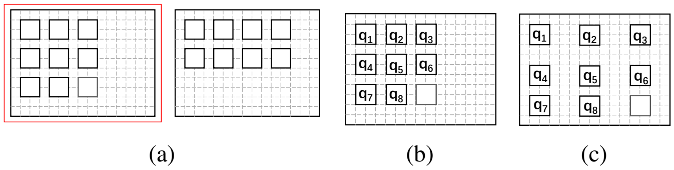


Figure 10: Tile location mapping process: (a) Shape determining, (b) mapping establishing, (c) bandwidth adjusting.

Algorithm 1: Scheduling for Limited Resources

```
   Input: Quantum Circuit $P$ and Chip $L_{m_1 \times m_2}$
   Output: Encoded Circuit $P^S$
1  tile_array = Tile_shaping($L_{m_1 \times m_2}$);
2  Mappings = Metis($G_C$);
3  M_location = Select(Mappings,cost function);
4  if double defect model then
5      M_cut = bipartite($G_P$);
6  end
7  while $G_P$ not empty do
8      gates=$G_P$.front_gate
9      gates_pri = priority(gates);
10     for $g_i(q_a, q_b) \leftarrow gates\_pri.begin()$ to
        $gates\_pri.end()$ do
11         if $Cut_a \neq Cut_b$ or model = lattice surgery then
12             $P^S$.add(path(gate, chip_now));
13         else
14             $M_a = M_{ta} + \theta M_{sa}$;
15             $M_b = M_{tb} + \theta M_{sb}$;
16             $min\_value, min\_index = \min(M_a, M_b)$;
17             if $min\_value < 0$ then
18                 $P^S$.add($Cut_{min\_index}$ modification);
19             else
20                 $P^S$.add(path(gate, chip_now));
21             end
22         end
23     end
24 end
25 return $P^S$, M_location
```

nodes, effectively reducing latency at the bottleneck (Line 7 - 12 in Algorithm 1). The priority of a gate is determined by the remaining gates number (how many gates depend on it) and criticality (the length of the critical path of the remaining gates). Gates with higher criticality are prioritized. When two CNOT gates have the same criticality, we select the gate with more remaining gates to allow more gates to execute earlier to utilize non-congested cycles better.

The time complexity of this algorithm is $O(g * m_1 * m_2)$, where g is the number of CNOT gates in the quantum circuit. The algorithm searches for paths for at most $O(g)$ gates, and the maximum time required to find a path for each CNOT gate is $O(m_1 * m_2)$. Here, $O(m_1 * m_2)$ is the number of nodes available for path selection on the chip, which is $m_1 * m_2 / d * d$, where d is the code distance and the side length of each tile.

Algorithm 2: Scheduling for Sufficient Resources

```
Input: Execution Scheme $E$
Output: Encoded Circuit $P^S$, initialization
now_step = 0;
while $i < E.length()$ do
    while $G$ is bipartite graph do
        for $gate \leftarrow E[i].begin()$ to $E[i].end()$ do
            G.add_edge(gate);
        end
        i++;
        $M_c$ = bipartite($G$);
    end
    if having mapping then
        $P^S$.add(change mapping to $M_c$);
    else
        initialization = $M_c$
    end
    for $j \leftarrow now\_step$ to $i$ do
        find braiding path($E[j]$);
        $P^S$.add($E[j]$);
    end
    now_step = i;
end
return $P^S$, initialization
```

Scheduling for Sufficient Resources. When $\lfloor \frac{b-1}{2} \rfloor + 3 \geq \widetilde{\mathbb{PM}}$, an execution scheme can be rapidly derived from Algorithm Para-Finding and Theorem 2. Algorithm Para-Finding provides $\widetilde{\mathbb{PM}}$ for this quantum circuit and a CNOT gate order scheme that achieves $\widetilde{\mathbb{PM}}$. This scheme indicates which gates are executed in each time cycle. Since the number of gates executed in each cycle is smaller than $\lfloor \frac{b-1}{2} \rfloor + 3$, employing the methods outlined in Proof 2 to determine the corresponding paths for these gates becomes feasible.

C. Optimizations for Double Defect Model

Previous works Braidflash [19] and Autobraid [17] do not consider the cut type by assuming all tiles have the same cut type. However, cut type is critical in transforming for double defect model, providing a significant opportunity to reduce the time on the table. For a CNOT gate, it takes **three** cycles to be executed if two involved tiles are of the same cut type, but only **one** cycle when cut types are different.

1) Cut Type Initialization:

The goal of the cut type initialization is to enable the execution of as many CNOT gates as possible within a single cycle. If the qubit communication graph is bipartite, we assign the same cut type to the logical qubits in the same set. This is the optimal cut type initialization, with which all CNOT gates can be implemented in one cycle.

However, for circuits whose qubit communication graph is not bipartite, find the optimal cut type initialization is NP-hard, according to Theorem 1. We propose a greedy algorithm that satisfies the requirement of cut type for gate executed earlier. To end this, firstly, we construct a sub-graph of the qubit communication graph where each vertex corresponds to a logical qubit. Then, we add the gates with no precursor in the current dag into the sub-graph. Next, we remove these gates in the DAG. Repeat these two steps until the newly added edges make the sub-graph no longer bipartite. The logical qubits belonging to the same set in this bipartite sub-graph are initialized to the same cut type.

2) Scheduling: When involving two tiles of a CNOT gate are of the same cut type, we estimate the impact of modifying cut type by calculating the M-value of each tile, specifically M-value $= M_t + \theta \times M_s$. M_t is the impact on time. It takes three cycles to execute the operation directly and four cycles with modification. If this tile is idle previously, the modification operation can be performed earlier to reduce the time cost. M_s is the impact of the occupation of the channel. CNOT gate needs two braiding operations between the tiles without changing the cut type but only needs one after modification. We adopt the look-forward strategy considering the impact of this modification on the children gates of this gate. The parameter θ is used to determine the weights of the two factors, M_t and M_s, in the current situation, where $\theta = (|\text{ready gate}| \times 2)/\text{bandwidth} \times n$. We choose to modify the type of the tile when the M-value is greater than 0 (Line 14 - 23 in Algorithm 1).

3) Sufficient Scheduling: When physical qubit resources are sufficient, we adopt the methods in Section IV-B2 to determine the tile location mapping and gate schedule scheme. The key idea for cut type initialization and scheduling is to make all CNOT gates execute in one cycle by remapping the cut type.

We propose the cut type scheduling algorithm Algorithm 2, whose execution flow is as follows. Firstly, we construct the qubit communication graph by sequentially adding edges from the execution scheme until it is no longer bipartite. Then, we use this bi-partition graph to initialize the cut type for executing this sub-circuit. When the operand tiles of CNOT gate are of the same cut type, our methods spend three cycles to modify the cut type to the new mapping found in the same way above. These two steps are iterated over until all the gates have been scheduled. We provide the cut type scheduling algorithm with $\frac{5}{2}$-approximation guarantee (as shown in Theorem 3).

Lemma 1: The qubit communication graph generated by any two layers of gates is bipartite.

Proof: Since logical qubits can participate in at most one CNOT gate in each layer, the qubit communication sub-graph generated by the 2-layer circuit has a maximum degree of two, and the two edges connected by a vertex must belong to two different layers. A graph with a maximum degree of two can only consist of lines or rings. A ring must be an even ring since two edges must be connected by a vertex in the odd ring that belongs to the same layer. As a result, this graph is bipartite since the qubit communication sub-graph can only consist of lines and even rings.

Theorem 3: Algorithm 2 is $\frac{5}{2}$-approximation.

Proof: For every two cycles of gates in the execution scheme given by Algorithm Para-Finding, the optimal cases must take two braiding cycles to execute since the gates with gate dependencies cannot be executed simultaneously. According to Lemma 1, our method requires at most five braiding cycles to execute these two layers of gates, three cycles for modifying to optimal cut type mapping, and two cycles for performing braiding operations. Thus, our algorithm is $\frac{5}{2}$-approximation.

V. Performance Evaluation

In this section, we first compare the performance of our methods to several state-of-the-art methods, AutoBraid [17] for double defect model and EDPCI [3] for lattice surgery model. Then we evaluate the performance of `Ecmas` as the communication resources increased. The details of our evaluation results are shown in Section V-B and we highlight our key findings as follows:

- For double defect model, `Ecmas` outperforms Autobraid [17], reducing the cycle of the transformed circuit by 67.3% at most, on average 51.5%.
- For lattice surgery model, `Ecmas` reaches the optimal solution in most of the test benchmarks, reducing the cycle of the transformed circuit by 13.9% at most compared with EDPCI [3].
- Compared with the result in the minimum viable chip, when the chip size increases 4x, `Ecmas` reduces the execution time by 10.8% and 30.9% in the double defect model and lattice surgery model respectively.
- `Ecmas` exhibits excellent scalability, effectively reducing the execution time of circuits as the chip size increases, while maintaining linear growth in compilation time.

A. Evaluation Setting

Metrics. We use the number of cycles to represent the communication time, which is used to measure the effectiveness of the compilation results.

Baselines. For double defect and lattice surgery models, we select the state-of-the-art algorithms AutoBraid [17] and EDPCI [3] as our baselines.

Chip Configuration. We evaluate our mapping and scheduling algorithm on three resource configurations $L_{l \times l}$: minimum viable, 4x, and sufficient qubits. For minimum viable qubits, $l = \lceil\sqrt{n}\rceil \times 5d$ for double defect model and $l = \lceil\sqrt{n}\rceil \times \lceil\sqrt{2}d\rceil$ for lattice surgery model, which is the smallest square grid chip that provides enough qubits. The 4x resource for lattice surgery is a chip with $l = \lceil\sqrt{n}\rceil \times 5d$. And the l for `Ecmas`-ReSu on sufficient resources depends on $\mathbb{PM}$ of the circuit.

Benchmarks. We use the quantum circuit from the previous works, including IBM Qiskit [29], ScaffCC [20], QUEKO [35], QASMbench [24] and random circuits with certain parallelism degree.

Evaluation Platform. Our experiments are performed on Intel(R) Xeon(R) CPU 6248R 96vCores 3.00GHz, with 256GB DDR4 memory. The operating system is Ubuntu 20.04.

B. Experiment Results

1) Double Defect Model: We evaluate the performance of `Ecmas` with two chip configurations: 1) minimum viable

Table I: Overview of Experiment Results

Circuit	n	α	g^1	Autobraid	Ecmas-dd		EDPCI		Ecmas-ls	
				Min	Min	ReSu	Min	4X	Min	4X
dnn_n8	8	48	192	147	**48**	48	48	53	**48**	**48**
grover	9	110	132	330	**166**	140	110	110	**110**	**110**
qpe_n9	9	42	43	126	**70**	54	42	42	**42**	**42**
BV_10	10	5	5	15	**5**	5	5	5	**5**	**5**
QFT_10	10	93	105	279	**165**	96	93	93	**93**	**93**
adder_n10	10	55	65	165	**78**	82	55	56	**55**	**55**
ising_n10	10	20	90	60	**20**	20	20	20	24	**20**
sat_n11	11	204	252	612	**336**	339	204	204	**204**	**204**
square_root_n4	11	221	294	663	**379**	389	221	225	**221**	**221**
multiplier_n15	15	133	222	399	**232**	244	133	134	**133**	**133**
qf21_n15	15	112	115	336	**197**	130	112	112	**112**	**112**
dnn_n16	16	48	384	198	**71**	48	79	53	**68**	**52**
square_root_n18	18	644	898	1932	**1047**	1133	644	645	**644**	**644**
ghz_state_n23	23	22	22	66	**22**	22	22	22	**22**	**22**
multiplier_n25	25	381	670	1143	**659**	717	383	385	**381**	**381**
swap_test_n25	25	63	96	201	**89**	99	67	65	**63**	**63**
wstate_n27	27	28	52	84	**28**	28	28	28	**28**	**28**
BV_50	50	27	27	81	**27**	27	27	27	**27**	**27**
QFT_50	50	2363	2435	7089	**4633**	2366	2363	2363	**2363**	**2363**
ising_n50	50	4	98	15	**10**	4	6	6	9	7
quantum_walk	11	14104	14372	42312	**20188**	19669	14104	14104	**14104**	**14104**
shor	12	13412	13838	40248	**22978**	20315	13412	13414	13414	**13412**

[1] n is the number of the qubits, α is the depth of the circuit, g is the CNOT gate of the circuit.

chip, and 2) sufficient resources. As shown in Table I, Ecmas outperforms AutoBraid methods with a reduction by 67.3% at most in the number of cycles, on average 51.5%. In Ecmas-ReSu, 40.9% of the circuits are further reduced in the number of cycles. Compared with AutoBraid, the average cycles are reduced by 57.1%. Increasing communication resources addresses the latency caused by braiding congestion. Thus, only the circuits suffering braiding congestion can benefit from the increase in bandwidth. The greater the parallelism, the more the time decreases as the bandwidth increases. Ecmas-ReSu is not always the best among these results. The reason is that the Ecmas-ReSu schedules gates have more strict limits for performance guarantee.

2) Lattice Surgery: For most circuits, our approach obtains the optimal solution as well as EDPCI in Table I. For circuits with higher $\mathbb{PM}$, Ecmas achieving better results reduction cycle of up to 13.9% compared with EDPCI. Since the Ecmas-ReSu for the lattice surgery model is guaranteed to yield the optimal solution (as described in Section IV-B2), we did not evaluate its performance here. Due to the absence of specialized optimizations in our approach for circuits with specific patterns, it falls EDPCI on the circuit ising_n10 on the minimum viable chip, whose CNOT gates are adjacent in the snake mapping. However, the absence of an effective initial mapping hampers EDPCI capacity to capitalize on the increased physical qubit resources. Our approach can leverage additional chip resources to reduce circuit depth. All results on the 4x resources are superior to or equal to the minimal viable chip.

C. Sensitivity Study

In this section, we conduct sensitivity studies to investigate the impact of our strategies, i.e., location and cut type initialization, gate prioritizing, and cut type scheduling. We also analyze the scalability of circuit parallelism and chip size. These results demonstrate that our method outperforms baselines on most quantum application circuits, especially those with medium to high parallelism. Moreover, our algorithm effectively utilizes the redundant physical qubit resources on the chip to reduce the circuit cycle.

1) Initialization Method: We examine the impact of initialization on the execution time in location and cut type. These evaluations and the subsequent scheduling experiments are conducted on the minimum viable chip.

Location: As illustrated in Table II, our method consistently shows superior performance in most circuits. We compared the influence of tile location selection on circuit depth with the trivial mapping in EDPCI [3] and the Metis mapping [21]. The trivial method refers to a twisting layout of logical qubits, where the qubits in the first row are placed from left to right, followed by the qubits in the second row placed from right to left, and repeated until all logical qubits are fully mapped. The shortcomings on the Ising circuit primarily result from the absence of specialized optimization targeting specific patterns. Nevertheless, the overall performance trend underscores our approach's robustness across diverse scenarios.

Cut type: In most cases, our method outperforms the baseline methods, as shown in Table III. We compare our cut type initialization algorithm with the random and max-cut algorithms. The random method assigns tiles with a random cut type.

Table II: Comparision of location initialization methods

Circuit name	n	α	g	Trivial	Metis	Ours
dnn_n8	8	192	48	48	72	**48**
grover	9	132	110	112	110	**110**
qpe_n9	9	42	43	42	42	**42**
ising_n10	10	90	20	20	36	**20**
adder_n10	10	65	55	55	55	**55**
QFT_10	10	105	93	95	93	**93**
multiply_n13	13	40	23	25	24	**23**
square_root_n18	18	898	644	644	644	**644**
ghz_state_n23	23	22	22	22	22	**22**
swap_test_n25	25	63	96	63	63	**63**
ising_n50	50	98	4	**4**	11	9

The max-cut method maximizes the number of CNOT gates with different cut types. We use the one_exchange method in NetworkX to implement the max-cut partition. For specific circuits such as ghz_state_n23, our initialization algorithm can significantly reduce the number of cycles. This is because the max-cut method aims to reduce the overall number of CNOT gates with different cut types. However, the cut type of tiles is dynamic since it can be modified during the execution. The initialization method should emphasize the front part of the quantum circuit.

Table III: Comparision of cut type initialization methods

Circuit name	n	α	g	Random	Max-cut	Ours
dnn_n8	8	48	192	64	48	**48**
grover	9	110	132	173	172	**166**
qpe_n9	9	42	43	73	76	**70**
ising_n10	10	20	90	37	29	**20**
adder_n10	10	55	65	85	82	**78**
QFT_10	10	93	105	171	173	**165**
multiply_n13	13	23	40	39	37	**35**
square_root_n18	18	644	898	1052	1053	**1047**
ghz_state_n23	23	22	22	48	40	**22**
swap_test_n25	25	63	96	120	94	**89**
ising_n50	50	4	98	11	10	**10**

2) Scheduling Strategy: We investigate our methods from two perspectives: gate scheduling and cut-type scheduling.

Gate scheduling: According to the results in Table IV, our method achieves optimal solutions in most benchmarks. We compare our gate scheduling method with the circuit-order approach in lattice surgery model, where circuit-order denotes scheduling gates based on their appearance in the circuit. Compared with circuit-order, our method optimizes up to 23% of the execution time.

Cut type scheduling: As shown in Table V, our algorithm outperforms the best baseline strategies on these benchmarks, achieving an average reduction of 25% and up to a maximum of 50%. We compared the cycle number of our methods with the other two strategies: Time-first and Channel-first. These two strategies determine whether to modify the cut type when dealing with a CNOT gate with different cut types. The former chooses the operations that make the CNOT gate complete as soon as possible, while the latter minimizes the channel occupation of this CNOT gate. Our optimization is caused by our strategy of adaptively adjusting the weights of time and channel based on resource conditions, making our strategy perform well in most scenarios.

Table IV: Comparison of different gate scheduling algorithms

Circuit name	n	α	g	Circuit-order	Ours
dnn_n8	8	48	192	66	**54**
grover	9	110	132	112	114
qpe_n9	9	42	43	42	**42**
ising_n10	10	20	90	26	**20**
adder_n10	10	55	65	55	**55**
QFT_10	10	93	105	93	**93**
multiply_n13	13	23	40	24	**23**
square_root_n18	18	644	898	644	**644**
ghz_state_n23	23	22	22	22	**22**
swap_test_n25	25	63	96	63	**63**
ising_n50	50	4	98	9	**9**

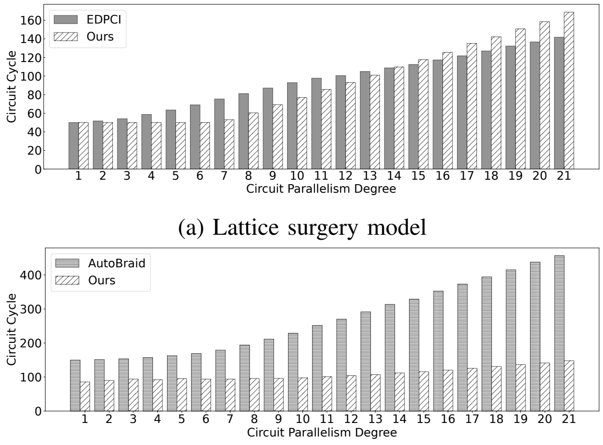
Table V: Comparison of different cut type scheduling

Circuit name	n	α	g	Channel-first	Time-first	Ours
dnn_n8	8	48	192	48	48	**48**
grover	9	110	132	166	174	**110**
qpe_n9	9	42	43	70	96	**42**
ising_n10	10	20	90	20	20	**20**
adder_n10	10	55	65	78	88	**55**
QFT_10	10	93	105	165	120	**93**
multiply_n13	13	23	40	35	41	**23**
square_root_n18	18	644	898	1047	1117	**644**
ghz_state_n23	23	22	22	22	22	22
swap_test_n25	25	63	96	89	102	**63**
ising_n50	50	4	98	8	8	**4**

3) Scalability: We explore the effectiveness of `Ecmas` on various input quantum circuits and chip sizes. Determining the parallelism of a given quantum circuit is challenging, but generating quantum circuits with specified parallelism is feasible. Inspired by QUEKO [35], we generate 50 random quantum circuits (as a test group) with 49 qubits, 50 depth, and parallelism ranging from 1 to 21. We use the average number of cycles in each group as the result.

Scalability of Circuit Parallelism Degree: In lattice surgery model, our approach generally outperforms the performance of EDPCI for most circuits, particularly in circuits with parallelism 3 to 13. Our method's performance is slightly less effective for circuits with high $\mathbb{PM}$ than that of EDPCI. This is due to our algorithm more likely to get trapped in local optima in these cases. In double defect model, the optimization ratio increased from 43% to 62.9% when Circuit Parallelism Degree increasing from 1 to 21, as shown in Fig. 11b. This is mainly attributed to our scheduling strategy for the cut type, which effectively leverages the waiting time due to path conflicts. We save significant channel resources by adjusting the cut type when the tile cut types are the same.

Scalability of Chip Size: Fig. 12 illustrates the trends of `Ecmas`'s performance (cycles) and efficiency (compiling time ratio) as the chip size increases. The compiling time ratio is $\tau_{(i,j)}/\tau_{(i,\min)}$ where $\tau_{(i,\min)}$ is compiling time of circuits with



(a) Lattice surgery model

(b) Double defect model

Figure 11: Effect of circuit parallelism

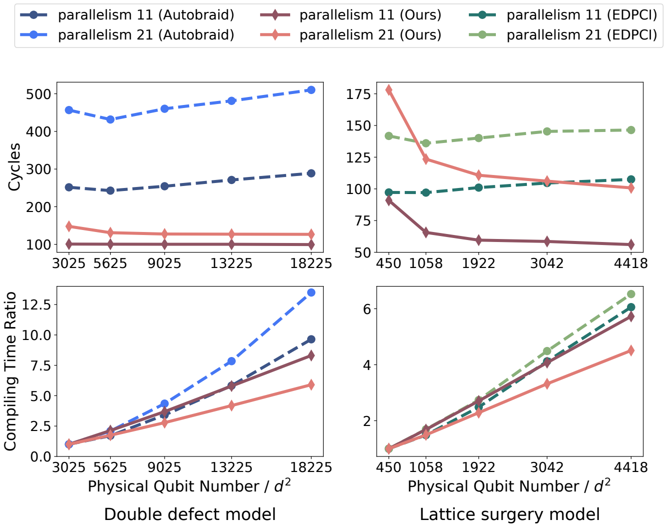


Figure 12: Effect of chip size

parallelism i at minimum viable chip and $\tau_{(i,j)}$ is compiling time of circuits with parallelism i at chip size j. We use this metric to fairly compare the scalability among the three algorithms Ecmas, Autobraid, and EDPCI were programmed in Python, C++, and Julia. The chip is a square with the average bandwidth per channel from 1 to 5. The result demonstrates that our methods' circuit cycles decrease as the chip size increases. The execution time of the circuits with $\mathbb{PM} = 21$ can be decreased by 10.8% for double defect model and decreased by 30.9% in lattice surgery model when the average bandwidth of the chip rises from 1 to 2.

VI. Related Work

Most existing quantum compilers [25], [33], [38], [40] focus on the physical qubit level compilation designed for NISQ circuits with 50 to 200 qubits, which is not fault-tolerant. These works focus on converting a logical circuit into a hardware-dependent physical circuit with respect to CNOT gates applied to physical qubits connected in the hardware.

Fault-tolerant compilation primarily focuses on architectures based on surface code, as it is the most promising error-correcting code in superconducting quantum computers. The fundamental difference between compiling a surface code circuit and a NISQ circuit is separating communication resources (channels) and computational resources (tiles). The resources are software-defined and can be specialized for specific circuits. The execution of CNOT gates is no longer achieved by moving the data to the two physically adjacent physical qubits. Instead, it takes place within the channel, using exclusive access to communication resources. Depending on the method of constructing logical qubits, surface code can be divided into double defect [10] and lattice surgery [16] with different logical operation implementation strategies. Double defect employs the braiding technique to perform CNOTgates. Braidflash [19] abstracts the constraints of CNOT gates implementation into braiding path disjoint. Autobraid [17] further discovers the local parallelism pattern and designs a stack-based search algorithm that enables efficient search for as many parallel CNOT gates as possible. Lattice surgery is a novel entrant in surface code approaches, employing a reduced number of physical qubits for encoding a logical qubit. It utilizes ZZ measurements for CNOT gate [26]. EDPCI [3] achieves long-range CNOT gates by utilizing ancilla tiles to construct Bell states. This approach requires a fourfold increase in physical qubits but enables the completion of CNOT gates at arbitrary distances within $2d$ surface code cycles. However, this approach does not account for the impact of initial mapping. Disregarding the circuit communication requirements with a trivial initial mapping results in a paradoxical situation where the circuit's performance worsens as chip resources increase.

Other works on fault-tolerant quantum compilation have focused on synergy with chip characteristics. Wu *et al.* [36] proposes a lattice encoding method for superconducting chips, which adapts the various chip structures to the surface code's 2D lattice. Previous works [6], [27] involve adapting the surface code to hexagonal chips, reducing chip connectivity to improve the accuracy of physical qubits. Some efforts [27] focus on utilizing operations with lower error rates during the compilation process to enhance circuit accuracy. Preskill *et al.* [28], on the other hand, centers on concatenating surface code and high-rate code, like quantum LDPC encoding, to address the challenges of low code rate and limited scalability in surface code implementations.

VII. Conclusion

In this paper, we study the surface code mapping and scheduling problem for the lattice surgery and double defect models. We formalize the problems in both models and establish the problem's complexity, particularly highlighting challenges in the double defect model. We introduce Circuit Parallelism Degree and Chip Communication Capacity to quantitatively analyze quantum circuits and quantum chips. Our mapping and scheduling methods, named Ecmas, feature algorithms for scenarios with sufficient and limited qubit resources. Extensive evaluations show that Ecmas provides significant reduction over the state-of-the-art approaches by reducing the execution time by 33.3% to 67.3% for double defect model and reducing by up to 13.9% for lattice surgery model.

Limitation and future work: As Circuit Parallelism Degree and Chip Communication Capacity are critical parameters for our mapping and scheduling methods, we still lack effective algorithms to obtain accurate results. In addition, our cost function to determine the importance of the current gate shows less effectiveness for high-parallel circuits compared to circuits with lower parallelism. Our investigation anticipates dynamic transforming strategies modifying mappings during the transforming process. Moreover, the complexity of scheduling problems and the bounds of chip communication capacity are still open problems.

ACKNOWLEDGEMENTS

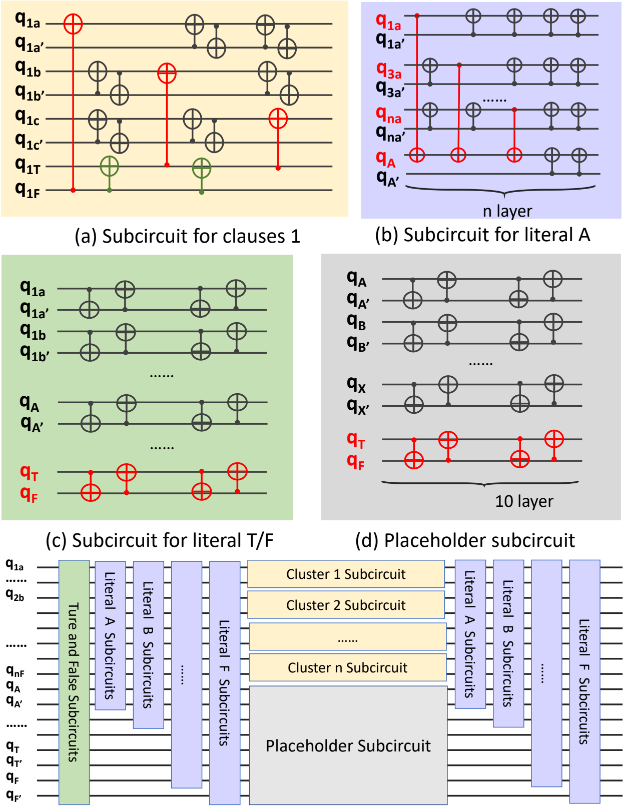
The research is partially supported by National Key R&D Program of China under Grant No.2021ZD0110400, Innovation Program for Quantum Science and Technology 2021ZD0302901, Anhui Initiative in Quantum Information Technologies under grant No. AHY150300 and China National Natural Science Foundation with No. 62132018, ”Pioneer” and ”Leading Goose” R&D Program of Zhejiang”, 2023C01029, and 2023C01143. Xiang-Yang Li is the corresponding author (Email: xiangyangli@ustc.edu.cn).

APPENDIX A
PROOF OF THEOREM 1

We prove that any instance of a 3-SAT problem can be reduced to an instance of a surface code initialization problem in polynomial time. We can construct the corresponding quantum circuit for any n-clause 3-SAT problem that the 3-SAT problem can be satisfied if and only if an initialization exists to execute this circuit no more than $10 + 3n$ cycles. Here, we assume that the bandwidth of the channel is sufficient. The quantum circuit, as shown in Fig.13e, is constructed in the following way:

For each three-literal clauses C_i construct a sub-circuit with 8 qubits, q_{ia}, q_{ib}, q_{ic} represent three literals a, b, c in the clause, $q_{ia'}, q_{ib'}, q_{ic'}$ are the ancilla qubits respectively, and q_{iT}, q_{iF} represent the logical qubits initialized into X-cut tile and Z-cut tile. If the clause's first literal a is positive, we add a CNOT gate between q_{1a} and q_{1T}, otherwise between q_{1a} and q_{1F}. Then we add a CNOT gate between q_{1T} and q_{1F}. After that we add two ancilla CNOT gates between q_{1b} and $q_{1b'}$, as well as q_{1c} and $q_{1c'}$. For the second and third literals b and c, the circuit is constructed in the same way as above. For example, the sub-circuits corresponding to clauses $(a \vee \neg b \vee c)$ are shown in Fig.13a. If the clause is true, a tail mapping exists, allowing the depth of this sub-circuit to be no more than 10. The cut type of each tile is mapped one-to-one with the true and false values of this literal. Here, the black gates are used for placeholders so that the tiles do not have time to change their tile type within 10 cycles.

Each three-literal clause generates a corresponding sub-circuit, which we connect in parallel, and the depth of it is no more than 10 if and only if all the clauses are True. Next, we must ensure that the same literal in different clauses corresponds to the same cut type. This is achieved through sub-circuit in Fig.13b. We declare an ideal literal and let the literal in different clauses perform CNOT gates with it. The circuit can achieve its shortest depth only if they are of different types to the ideal literal. Here, the black gate is used for placeholder operations if a tile modifies its tile type and to supplement the circuit so that the shortest depths corresponding to different literals are n. For the ideal True and False, we require an additional n-depth sub-circuit in Fig.13c that makes their cut types different. Sub-circuit Fig.13d ensures that the ideal literal does not modify its cut type while waiting for the clauses sub-circuit to execute.



(a) Subcircuit for clauses 1 (b) Subcircuit for literal A

(c) Subcircuit for literal T/F (d) Placeholder subcircuit

(e) Circuit corresponding to 3SAT problem

Figure 13: Quantum circuit construction for an n-clause 3-SAT Problem

REFERENCES

[1] “Suppressing quantum errors by scaling a surface code logical qubit,” *Nature*, vol. 614, no. 7949, pp. 676–681, 2023.

[2] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, “Quantum supremacy using a programmable superconducting processor,” *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[3] M. Beverland, V. Kliuchnikov, and E. Schoute, “Surface code compilation via edge-disjoint paths,” *PRX Quantum*, vol. 3, no. 2, p. 020342, 2022. [Online]. Available: https://github.com/eddieschoute/TeleportRouter.jl

[4] S. Bravyi and J. Haah, "Magic-state distillation with low overhead," *Physical Review A*, vol. 86, no. 5, p. 052329, 2012.

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

[6] C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross, "Topological and subsystem codes on low-degree graphs with flag qubits," *Physical Review X*, vol. 10, no. 1, p. 011022, 2020.

[7] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 2002.

[8] Y. Ding, A. Holmes, A. Javadi-Abhari, D. Franklin, M. Martonosi, and F. Chong, "Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 828–840.

[9] G. Finke, P. Lemaire, J.-M. Proth, and M. Queyranne, "Minimizing the number of machines for minimum length schedules," *European Journal of Operational Research*, vol. 199, no. 3, pp. 702–705, 2009.

[10] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A*, vol. 86, no. 3, p. 032324, 2012.

[11] I. M. Georgescu, S. Ashhab, and F. Nori, "Quantum simulation," *Reviews of Modern Physics*, vol. 86, no. 1, p. 153, 2014.

[12] C. Gidney and M. Ekerå, "How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, p. 433, 2021.

[13] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[14] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, "Supervised learning with quantum-enhanced feature spaces," *Nature*, vol. 567, no. 7747, pp. 209–212, 2019.

[15] D. Herr, F. Nori, and S. J. Devitt, "Optimization of lattice surgery is np-hard," *Npj quantum information*, vol. 3, no. 1, p. 35, 2017.

[16] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, "Surface code quantum computing by lattice surgery," *New Journal of Physics*, vol. 14, no. 12, p. 123011, 2012.

[17] F. Hua, Y. Chen, Y. Jin, C. Zhang, A. Hayes, Y. Zhang, and E. Z. Zhang, "Autobraid: A framework for enabling efficient surface code communication in quantum computing," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 925–936. [Online]. Available: https://github.com/huafei1137/Autobraid

[18] H.-Y. Huang, M. Broughton, M. Mohseni, R. Babbush, S. Boixo, H. Neven, and J. R. McClean, "Power of data in quantum machine learning," *Nature communications*, vol. 12, no. 1, pp. 1–9, 2021.

[19] 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*, 2017, pp. 692–705.

[20] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi, "Scaffcc: A framework for compilation and analysis of quantum computing programs," in *Proceedings of the 11th ACM Conference on Computing Frontiers*, 2014, pp. 1–10.

[21] G. Karypis, "Metis: Unstructured graph partitioning and sparse matrix ordering system," *Technical report*, 1997.

[22] A. Y. Kitaev, "Fault-tolerant quantum computation by anyons," *Annals of physics*, vol. 303, no. 1, pp. 2–30, 2003.

[23] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann *et al.*, "Realizing repeated quantum error correction in a distance-three surface code," *Nature*, vol. 605, no. 7911, pp. 669–674, 2022.

[24] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation," *ACM Transactions on Quantum Computing*, 2022.

[25] 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*, 2019, pp. 1001–1014.

[26] D. Litinski, "A game of surface codes: Large-scale quantum computing with lattice surgery," *Quantum*, vol. 3, p. 128, 2019.

[27] M. McEwen, D. Bacon, and C. Gidney, "Relaxing hardware requirements for surface code circuits using time-dynamics," *arXiv preprint arXiv:2302.02192*, 2023.

[28] C. A. Pattison, A. Krishna, and J. Preskill, "Hierarchical memories: Simulating quantum ldpc codes with local gates," *arXiv preprint arXiv:2303.04798*, 2023.

[29] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," 2023.

[30] M. Schuld and N. Killoran, "Quantum machine learning in feature hilbert spaces," *Physical review letters*, vol. 122, no. 4, p. 040504, 2019.

[31] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[32] P. W. Shor, "Fault-tolerant quantum computation," in *Proceedings of 37th Conference on Foundations of Computer Science*. IEEE, 1996, pp. 56–65.

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

[34] S. A. Stein, B. Baheri, D. Chen, Y. Mao, Q. Guan, A. Li, S. Xu, and C. Ding, "Quclassi: A hybrid deep neural network architecture based on quantum state fidelity," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 251–264, 2022.

[35] B. Tan and J. Cong, "Optimality study of existing quantum computing layout synthesis tools," *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1363–1373, 2020.

[36] A. Wu, G. Li, H. Zhang, G. G. Guerreschi, Y. Ding, and Y. Xie, "A synthesis framework for stitching surface code with superconducting quantum devices," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 337–350.

[37] Y. Wu, W.-S. Bao, S. Cao, F. Chen, M.-C. Chen, X. Chen, T.-H. Chung, H. Deng, Y. Du, D. Fan *et al.*, "Strong quantum computational advantage using a superconducting quantum processor," *Physical review letters*, vol. 127, no. 18, p. 180501, 2021.

[38] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen, and E. Z. Zhang, "Time-optimal qubit mapping," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 360–374.

[39] Y. Zhao, Y. Ye, H.-L. Huang, Y. Zhang, D. Wu, H. Guan, Q. Zhu, Z. Wei, T. He, S. Cao *et al.*, "Realization of an error-correcting surface code with superconducting qubits," p. 030501, 2022.

[40] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the ibm qx architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2018.