Ecmas+: Efficient Circuit Mapping and Scheduling for Surface Code Encoded Circuit on Quantum Cloud Platform

MINGZHENG ZHU, School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
HAO FU, School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
HAISHAN SONG, School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
JUN WU, School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
CHI ZHANG, Hefei University of Technology, Hefei, China and Engineering Research Center of Safety Critical Industrial Measurement and Control Technology, Ministry of Education, Hefei, China
WEI XIE, School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
XIANGYANG LI, Hefei National Laboratory, Hefei, China and School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

As the leading candidate for quantum error correction, the surface code faces substantial overhead, such as redundant physical qubits and prolonged execution time. Reducing the space-time cost of circuit execution can significantly improve the throughput of modern quantum cloud platforms. While utilizing more physical qubits can reduce execution time, different quantum circuits vary in their ability to leverage chip resources. Therefore, optimizing the compilation of surface code circuits on quantum chips becomes critical. In this work, we address the mapping and scheduling problem in compiling surface code to reduce the cost. First, we introduce a novel metric Circuit Parallelism Degree to characterize circuit properties in detail and select the most suitable chip from a list of available options. Next, we will quantitatively assess the resources to determine if they are sufficient for the circuit. We then propose a resource-adaptive mapping and scheduling method called **Ecmas+**, which customizes the initialization of chip resources for each circuit. **Ecmas+** significantly reduces execution time in the double defect and lattice surgery models. Extensive numerical tests on practical datasets demonstrate that **Ecmas+** outperforms state-of-the-art methods, reducing execution time by an average of 46% for the double defect model and 29.7% for the lattice surgery model.

Authors' Contact Information: Mingzheng Zhu, School of Computer Science and Technology, University of Science and Technology of China, Hefei, China; e-mail: zmzming@mail.ustc.edu.cn; Hao Fu, School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China; e-mail: hflash@mail.ustc.edu.cn; Haishan Song, School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China; e-mail: haishansong@mail.ustc.edu.cn; Jun Wu, School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China; e-mail: jun_wu@mail.ustc.edu.cn; Chi Zhang, Hefei University of Technology, Hefei, Anhui, China and Engineering Research Center of Safety Critical Industrial Measurement and Control Technology, Ministry of Education, Hefei, Anhui, China; e-mail: zhangchi@hfut.edu.cn; Wei Xie, School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China; e-mail: xxieww@ustc.edu.cn; Xiangyang Li, Hefei National Laboratory, Hefei, Anhui, China and School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China; e-mail: xiangyangli@ustc.edu.cn.

ACM 1544-3973/2025/8-ART
https://doi.org/10.1145/3760783

Additional Key Words and Phrases: Circuit Mapping and Scheduling, Surface Code, Quantum Computing

1 Introduction

Quantum algorithms offer exponential speedup over classical algorithms in various fields such as machine learning [18, 22, 39], quantum simulation [15] and cryptography [40]. However, these advantages are hard to realize due to the inherent fragility of quantum computing hardware. The error rate of the state-of-the-art superconducting quantum devices is around 10^{-3} per operation [3, 17, 45], which falls far short of meeting the demands of practical quantum applications [16, 38]. To counteract these errors, implementing quantum error correction (QEC) is essential, paving the way to the framework for fault-tolerant computation [41].

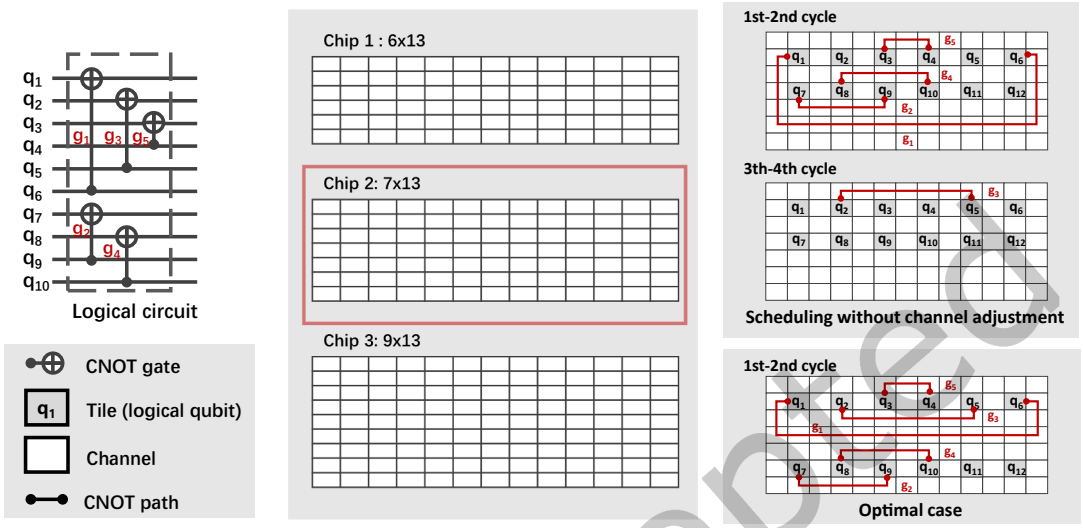
However, QEC is highly resource-intensive [6, 35] and is estimated to consume up to 90% of the resources. For example, surface code [5, 8, 27], the most widely used QEC code, requires a large number of physical qubits to encode a single logical qubit, referred to as a **tile** [23]. In addition, numerous ancilla tiles are needed to perform logical operations. Executing a CNOT gate requires ancilla tiles to provide the communication resources [12, 21]. Likewise, implementing a T gate demands both communication resources and ancilla tiles for magic state preparation [10, 32]. If communication resources are sufficient, the circuit can be executed efficiently. Otherwise, communication congestion occurs, leading to delays and waiting times [49]. The qubit resource requirements of circuits can vary significantly depending on their operation patterns. In circuits with sparse operations, where only a few operations are executed simultaneously, increasing qubit resources may not reduce execution time. In contrast, adding communication resources can effectively minimize delays for circuits with dense operations.

Nowadays, the mainstream way to use quantum computers is through quantum cloud services [9, 24], primarily due to quantum devices' high prices and maintenance complexity. To provide efficient quantum services in the cloud, optimizing compilation on a given chip is not enough. It is also essential to select the most suitable chip for a given circuit. This integrated workflow ensures better qubit utilization, significantly boosting the overall throughput of the quantum cloud platform.

Therefore, the quantum circuit execution process on the quantum cloud can be divided into two steps: selecting an appropriate chip and transforming the circuit on the chip. During the chip selection phase, the selector analyzes the circuit's characteristics and selects the most suitable chip, minimizing the cost of executing the given circuit. We define this cost as **Circuit Execution Cost**, the time-space cost of the circuit's execution. Once an appropriate chip is selected, the transformer aims to reduce the circuit's execution time. The transforming phase can be further divided into initialization and scheduling. In initialization, the transformer assigns tiles to each logical qubit and establishes communication channels. In scheduling, the transformer determines the precise execution plans for each operation. Most operations can be performed within the tiles [12], except the T and CNOT gates. The overhead of preparing magic states has been considerably reduced through extensive research efforts [10, 32]. However, communication-induced delays can be substantial, significantly degrading both the fidelity of execution results and overall system throughput. Communication, specifically the execution of 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 [12] or lattice surgery model [4]. 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 two cycles regardless of the path length. Simultaneous execution of CNOT gates requires non-intersect corresponding paths.

Many works [4, 21, 23] focus on building CNOT gate paths to reduce the latency caused by path conflicts. However, they overlooked a crucial aspect: In surface code encoding, the function of tiles is software-defined, whether they are used as logical tiles or communication resources. The transformer can customize the initialization of chip resources to reduce circuit execution time. Moreover, different circuits require different resources. A chip

selector is needed to assign the suitable chip for each circuit for quantum cloud platforms with multiple types of chips.



(a) logical circuit (b) chips list (c) transforming results

Fig. 1. Motivation example: a logical quantum circuit and its corresponding surface code encoded circuit. In this example, we choose to execute the circuit on Chip 2, where the execution can be completed in 2 cycles.

Motivation Example: As illustrated in Fig. 1a, five independent gates are ready for execution. Each gate's execution path requires a specific width of physical qubits. There are three different types of chips available (Fig.1b). For Chip 1, the circuit requires a minimum of four cycles to execute. In contrast, for Chip 2, the circuit can be completed in 2 cycles under optimal scheduling, with better allocation of communication resources. Although Chip 3 offers more qubit resources, the demand for resources is already saturated, resulting in no reduction in execution time. Therefore, Chip 2 is selected, with resource-customized initialization considered during scheduling.

In this paper, we address the problems of chip selection and surface code circuit mapping and scheduling. We first characterize circuit features and select the appropriate chip based on these characteristics. Next, we propose a chip selector and a resource-adaptive transformer, **Ecmas+**, which minimizes circuit execution cost and executing time. Our contributions are summarized as follows:

- We formulate the chip selection problem for surface code execution on multi-chip quantum systems and formulate surface code circuit mapping and scheduling problems in double defect and lattice surgery models.
- We define two novel metrics to characterize both circuit demands and chip capacities and further propose the *Circuit Execution Cost* metric to capture the time-space trade-off of a quantum circuit.
- We present a resource-adaptive compilation framework **Ecmas+** composed of:
  - A **chip selector**, which evaluates candidate chips and selects a cost-effective one based on circuit features.
  - A **circuit transformer**, which customizes chip resource initialization and extends compilation to a universal gate set (CNOT, H, T, and Pauli).

  With sufficient physical qubits, **Ecmas+**-ReSu offers efficient, performance-guaranteed compilation results.

- We evaluate **Ecmas+** for circuits from IBM Qiskit [36], QASMbench [30], etc. **Ecmas+** eliminates 46% of the execution time on average compared with Autobraid [21] for the double defect model. **Ecmas+** could find the optimal result for most benchmarks for the lattice surgery model. Compared with EDPCI [4], **Ecmas+** can achieve optimizations on average 29.7%.

The subsequent sections of this paper are arranged as follows. Section 2 lays the groundwork with an introduction to the necessary background. We then formulate the problem of chip selection and surface code mapping and scheduling in Section 3. Our methodologies are presented in Section 4, followed by an evaluation of their performance in Section 5. Related work is reviewed in Section 6, followed by discussion and conclusion in Sections 7 and 8.

2 Background

2.1 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 fundamental units in quantum computing that can be represented by a normalized vector. Quantum gates are unitary operations that operate on qubits. A universal gate set, such as X, Y, Z, H, T, and CNOT gates, can combine arbitrary quantum circuits.

However, quantum computers are inherently susceptible to noise from environmental interactions and to inaccuracies in operations. Thus, quantum error correction (QEC) codes are essential for constructing fault-tolerant quantum computing systems. These codes enhance information redundancy by encoding a logical qubit into multiple physical qubits, thereby increasing reliability. The noise of the quantum system appears not only in the communication process but also in the computation process. Hence, the reliability of quantum circuits is heavily dependent on the safeguarding offered by QEC. It is essential for QEC protocols to detect and correct errors periodically during the execution.

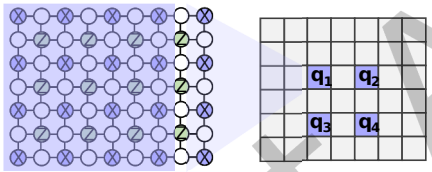
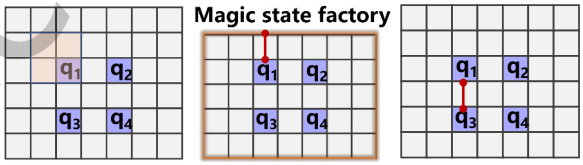


Fig. 2. Surface code model: physical qubits and simplified representation of chip.



(a) H gate (b) T gate (c) CNOT gate

Fig. 3. Surface code encoded operations in the lattice surgery model.

2.2 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 [2, 28, 48].

As shown in Fig. 2, surface code is realized on a 2D lattice of physical qubits, including data and measurement qubits. Data qubits (the white circles) store quantum states, while measurement qubits (the purple and green circles) identify error occurrences. Physical qubits can be divided into computation and communication resources at the logical layer. The computation resources are the physical qubits, which are encoded as logical qubits and simplified into small boxes. The remaining qubits on the chip are ancilla qubits utilized to construct paths for executing the CNOT gate communication.

Surface code can be classified as the double defect model [12] and lattice surgery model [4] based on the different approaches to creating logical qubits. During the execution, measurement circuits are periodically

executed to detect errors. The time for executing one measurement circuit is called a surface code cycle. Code distance d is an important parameter in QEC codes. As d increases, more physical qubits are used to encode a single logical qubit with lower error rates for each logical qubit. Execute a quantum circuit under surface code protection, requiring all logical gates to be transformed into their surface code-encoded forms.

Pauli gates: Pauli gates are X, Y, and Z gates, which can be executed inside the tiles within one surface code cycle. In practice, these gates are usually handled entirely by the classical control software [12] rather than physically implemented on a quantum computer, reducing the noise and execution cost that physical operations introduce. Given that the execution time of these gates is constant with respect to the code distance d, and considering that these gates are predominantly executed in software, we will ignore their execution times in subsequent comparisons of results.

H gate: The H gate can be executed locally, only involving the surrounding ancilla tiles. Different operational schemes may exhibit subtle differences in execution time and the extent of the tiles involved. In this paper, we proceed with the H gate as follows. The lattice surgery model requires 2×2 tiles and three cycles, that is, $3d$ surface code cycles, as shown in Fig. 3a. During the whole compilation procedure, parameter d remains consistent; hence, we designate cycle as the unit of execution time, with the definition of one cycle being equivalent to d surface code cycles. The H gate can be executed inside the tile with 3 cycles for the double defect model.

T gate: The execution of the T gate consists of two parts: preparing magic states through a magic state factory and executing the magic state injection circuit (CNOT gate) between the prepared magic states and the involved tile. This paper assumes an abundance of magic states surrounding the chip, as shown in Fig. 3b. At any given moment, we can access a sufficient quantity of magic states at the edge of the chip.

CNOT gate: CNOT gates are implemented differently across encoding models. For the lattice surgery model, performing a CNOT gate means finding a Bell state path between the tiles corresponding to two qubits [4]. As shown in Fig. 3c, paths of arbitrary length require two cycles, and paths executed within the same cycle cannot intersect. For the double defect model, executing a CNOT gate requires finding non-intersecting paths between tiles, and the execution time of the path is independent of its length. One distinction to be aware of is that each tile has an additional attribute called cut type. Each tile is initialized into one of two cut types: X-cut type or Z-cut type. Executing CNOT gates between tiles of different cut types requires two cycles, whereas executing CNOT gates between tiles of the same cut type requires six cycles. In addition, we can spend six cycles to change the cut type of each tile.

3 System Model and Problem Formulation

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

Quantum Circuit: We consider an input quantum circuit P with n logical qubits (Fig.4a). This circuit can be represented as a weighted directed acyclic graph $G_P = (V, E, W)$ (Fig.4b), where vertices represent quantum gates and edges denote gate dependencies. Each node v_i is assigned a weight that represents the gate execution time w_i in the surface code model. The critical path length of G_P is circuit depth, denoted by α. To better characterize the circuit depth under surface code encoding, we introduce surface code weighted circuit depth (α_s). This metric is defined as $\alpha_s = \max \sum_{i=0}^{k} w_i$, where w_i is the weight of node v_i along a path from the root to a leaf node, $path = (v_1, v_2, \ldots, v_k)$.

Quantum Chip: We assume the quantum chip adopts a 2D lattice topology, denoted as $L_{m_1 \times m_2}$—a grid of m_1 rows and m_2 columns of physical qubits. To decouple our framework from hardware-specific parameters (e.g., error rates), we abstract this lattice into a tile-level model. Each logical qubit is mapped to a tile, a square region of physical qubits, denoted as $T_{a,b}$ where (a, b) is its upper-left coordinate. Communication between tiles occurs via channels, where each lane represents a unit of bandwidth capable of supporting one independent CNOT path.

The number of lanes in a channel defines its bandwidth, and the chip bandwidth is the minimum bandwidth among all channels.
The physical size of each tile and the width of each lane depend on the encoding model. As illustrated in Fig. 5, for the double-defect model, each tile occupies a $5d \times 5d$ area, and each lane requires $2d$ physical qubits. Additionally, in this model, each tile is associated with a cut type. We use Cut_i to represent the cut type of tile T_i. In the lattice surgery model, the tile and lane dimensions are $\lceil\sqrt{2}d\rceil \times \lceil\sqrt{2}d\rceil$ and $\sqrt{2}d\rceil$, respectively. The parameter d is the code distance.

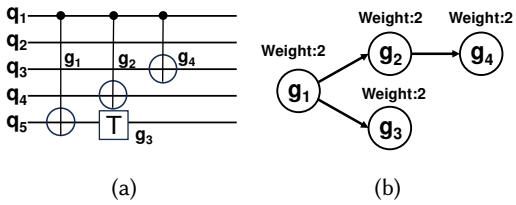


Fig. 4. Representations of the quantum circuit: (a) logical quantum circuit, (b) weighted DAG representation.

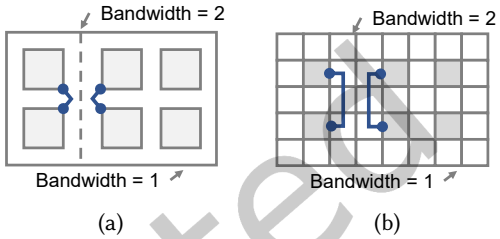


Fig. 5. Quantum Chip: (a) double defect model, (b) lattice surgery model.

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 the same cycle do not intersect. The execution time of a circuit is $\Delta \times d \times \tau$, 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 the circuit Δ_{PS} be minimized.

3.1 Chip Selection

Circuit Execution Cost: We define the time-space resources required to execute a quantum circuit on a given chip as its Circuit Execution Cost, which is the product of the number of qubits used and the circuit's minimum execution time. In this work, we assume that only one circuit can be executed on a chip at a time. Consequently, the qubit resources allocated to the quantum circuit are considered the total qubit resources available on the chip it occupies.

Problem 1. ***Chip Selection.***
Input: *An input logical circuit P, a code distance d and a list of available 2D lattice chips $L_1, L_2, ...L_n$.*
Output: *Chip L_i in chip list with which P has the minimal Circuit Execution Cost.*

3.2 Transformation in Double Defect Model

Problem 2. ***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 $\alpha_s + k$, namely, $\Delta^{P^S_{OPT}} < \alpha_s + k$.*

Problem 3. ***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: Zhu *et al.* [49] proved that the complexity of surface code mapping and transforming problem is NP-hard for the double defect model.

3.3 Transformation in Lattice Surgery Model

Problem 4. ***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 $\Delta^{P^S_{OPT}} < \alpha_s + k$, namely, the number of cycles of the optimal surface code encoded circuit P^S_{OPT} is upper bounded by $\alpha_s + k$.*

Problem 5. ***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.* [19] demonstrated that the complexity of surface code mapping and transforming problem is NP-complete for the lattice surgery model.

4 System Design

To improve the throughput of quantum cloud platforms, it is essential not only to optimize the execution time of circuits but also to strike a balance between the number of qubits used and the execution time. Our framework, **Ecmas+**, not only maximizes resource utilization on a given chip to minimize circuit execution time but also selects appropriate chips for circuits when multiple options are available on the quantum cloud platform. To achieve this, we carefully analyze circuit features and introduce a new metric, Circuit Execution Cost, to quantify the quantum resources consumed during circuit execution, which aids in evaluating chip selection quality. Once a chip is selected, our transformer assesses whether the available qubit resources meet the requirements based on the characteristics of both the circuit and the chip. Depending on resource availability, we design different algorithms to transform the circuits efficiently. When sufficient physical qubits are present on the chip, **Ecmas+**-ReSu can achieve shorter transformation times while ensuring performance guarantees. An overview of our comprehensive tool flow is shown in Fig. 6.

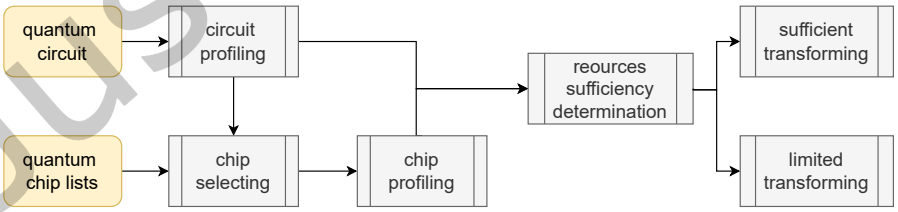


Fig. 6. Overview of **Ecmas+**.

4.1 Quantum Circuit Profiling

Quantum circuits vary in communication resource demands. To better meet the requirements of the circuit, we analyze its characteristics, focusing on the following key metrics:

(1) **Number of qubits n:** This metric denotes the total count of qubits present within the quantum circuit.
(2) **Logical circuit depth α** and **Surface code weighted circuit depth α_s:** These parameters estimate the minimum execution time necessary for the circuit's operation.
(3) **Rate of different gates:** This aspect encompasses the frequency and quantity of various types of quantum gates utilized within the quantum circuit, as well as the overall gate density. As an example, the CNOT gate rate is defined as $r_{cnot} = \frac{g_{cnot}}{n \times \alpha}$, where g_{cnot} represents the number of CNOT gates.
(4) **Peak communication demand:** We introduce *Circuit Parallelism Degree* (denoted as $\mathbb{PM}$) to characterize the maximum demand of communication resources of a circuit. This factor is a critical constraint in determining whether the execution of the quantum circuit can achieve an optimal solution. The definition of Circuit Parallelism Degree varies slightly depending on the model, with more specific definitions.

Definition 1. Gate partition π

For a quantum circuit $G_P = (V, E, W)$, we construct a new graph $G'_P = (V', E')$ by replicating each node v from the set V, w_v times, resulting in the nodes $v_1, v_2, \ldots, v_{w_i}$. These nodes are ordered such that $v_1 > v_2 > \cdots > v_{w_i}$. For each edge $(u, v) \in E$ from the graph G_P, the corresponding nodes in the new graph must satisfy the following condition: $u_{wu} > v_1$, respectively. Given the graph G'_P, a partition π is to divide the set V' into α_s disjoint sets $V_1, V_2, ..., V_{\alpha_s}$, such that for $u \in V_i$ and $v \in V_j$, if $(u, v) \in E'$, then $i > j$. And nodes $v_1, v_2, \ldots, v_{w_i}$, which are replicated from the same original node v, must be allocated to adjacent subsets. For example, if v_1 is in subset V_j, then $v_2, \ldots, v_{w_i}$ must be placed in subsequent subsets $V_{j+1}, \ldots, V_{j+w_i-1}$.

Definition 2. Circuit Parallelism Degree of double defect model

$\mathbb{PM} = \min_\pi \max_{i=1}^{\alpha_s} f_{cnot}(V_i) + f_t(V_i)$ *for all possible partition π , where $f_{cnot}(V_i)$ is the number of CNOT gate in set V_i and $f_T(V_i)$ is the number of T gate in set V_i.*

Definition 3. Circuit Parallelism Degree of lattice surgery model

$\mathbb{PM} = \min_\pi \max_{i=1}^{\alpha_s} length(V_i)$ *for all possible partition π , where $length(V_i)$ is the number of gate in set 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 *et al.* [11] has proved that this is NP-complete. We propose a heuristic Algorithm. 1 to find the circuit's estimated Circuit Parallelism Degree $\widetilde{\mathbb{PM}}$ and the execution schema $\mathcal{E}$. Our method uses layers to keep track of the execution order of gates, where $\mathcal{E}$[i] represents the operations to be performed in the i-th layer. For any gate i, we record two high and low values to denote the highest and lowest layers in which the gate can be scheduled. Layers are determined by the gate's parents and children nodes as shown in Line 4 - 9. Then, we calculate the difference value of each gate and choose the gate with the smallest difference value (Line 10 - 13). For this gate, we schedule it to the layer with the smaller f_{gate} to execute in all the possible layers. For double defect model, $f_{gate}(\mathcal{E}[i]) = f_{cnot}(\mathcal{E}[i]) + f_t(\mathcal{E}[i])$, and $f_{gate}(\mathcal{E}[i]) = length(\mathcal{E}[i])$ for lattice surgery model. After that, we update the low and high values of each gate. We repeat this process until all the gates are scheduled. The maximum $f_{gate}(\mathcal{E}[i])$ among all layers is $\widetilde{\mathbb{PM}}$ of this circuit.

4.2 Quantum Chip Selecting

The chip resources discussed here primarily focus on communication resources. We follow the settings from previous studies [4], where magic state factories are placed at the chip boundaries, and magic states are assumed to be abundant. Here, we use the lattice surgery model as an example. Since H-gates are executed within the tile in the double defect model, it is equivalent to the lattice surgery model with the same number of CNOT and T gates but no H-gates.

Finding the trade-off between the qubit resources and the execution time is not straightforward. To address this challenge efficiently, first, we analyze the relationship between circuit depth α, gate rates r, chip size, and

Algorithm 1: Circuit Profiling

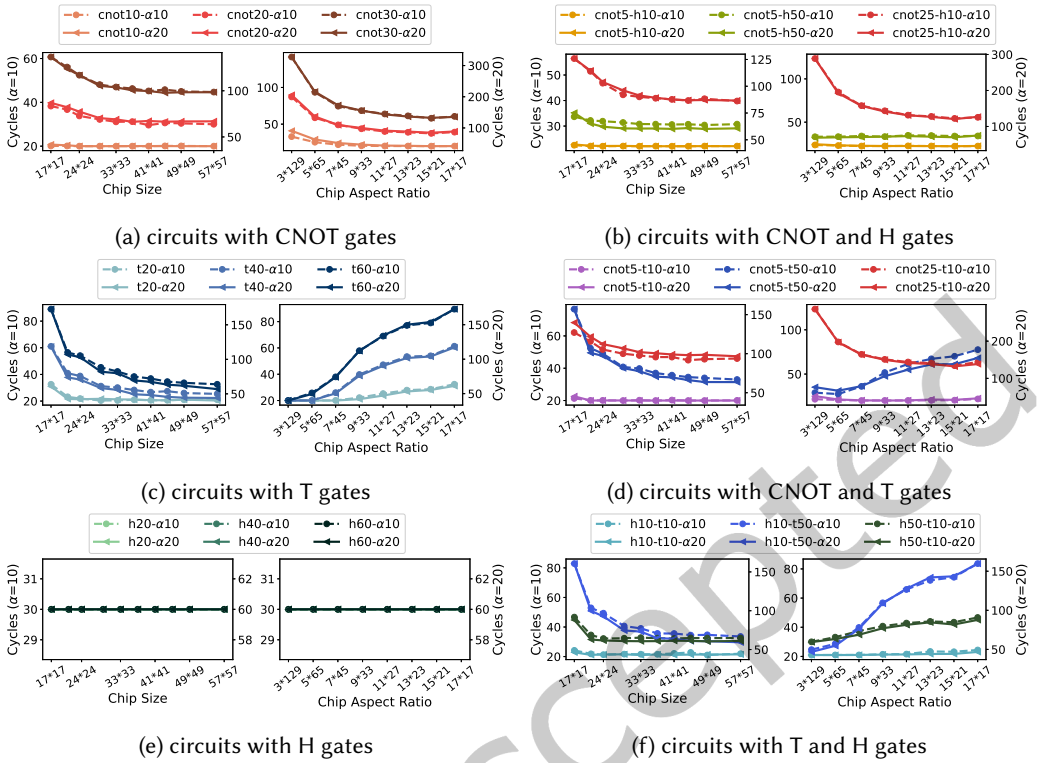
Input: Quantum Circuit P;
Output: Circuit Parallelism Degree $\widetilde{\mathbb{PM}}$ and Execution Scheme $\mathcal{E}$;

$\mathcal{E}$ = [] * Δ_{G_P};
G_u = unscheduled gates;
while $G_u \neq \emptyset$ **do**
    high[i], low[i], difference[i] = [0]×$P.gate\_num$;
    for $gate_i \leftarrow gates\_topo.begin()$ **to** $gates\_topo.end()$ **do**
        low[i] = max (low[gate$_i$.parents] + gate$_i$.parents.weight);
    for $gate_i \leftarrow gates\_topo.end()$ **to** $gates\_topo.begin()$ **do**
        high[i] = max (high[gate$_i$.children] + gate$_i$.children.weight);
        difference[i] = Δ_{G_P} - low[i] - high[i];
    Find gate$_i$ with minimal difference;
    Find Layer j with minimal $\sum_{j}^{j+gate_i.time} f_{gate}(\mathcal{E}[j])$;
    for *index in range(gate$_i$.weight)* **do**
        $\mathcal{E}$[j+index].append(i);
    G_u delete gate$_i$
$\widetilde{\mathbb{PM}} = max_{i=0}^{\Delta_{G_P}} f_{gate}(\mathcal{E}[i])$;
return $\widetilde{\mathbb{PM}}$ and $\mathcal{E}$;

the chip aspect ratio. To explore how these factors influence execution time, we generated 20 random quantum circuits for each set of circuit characteristics. Each circuit consists of 64 qubits. For instance, a circuit named "cnot5-h5-α10" refers to a random circuit with a depth of 10, where each layer contains, on average, 5 CNOT gates and 5 H gates. By transforming these circuits on different chip configurations, we aim to observe trends in execution time based on varying circuit depths, gate densities, and chip properties. As shown in Fig. 7, the results indicate the following key patterns:

(1) **Logical depth independence.** The solid and dashed lines in each graph represent the same trend, showing the average results for circuits with depths of 10 and 20, respectively.
(2) **T-gates perimeter-sensitive.** When the T-gates are dense, increasing the chip's perimeter rapidly reduces the circuit's execution time. The main reason is that we assume magic states are obtained from the chip's edge, so the perimeter directly impacts the efficiency of accessing them.
(3) **CNOT gates size sensitivity.** When CNOT gates are dense, the execution time decreases with increased chip size but increases as the chip becomes more elongated.
(4) **H gate smoothing effect.** In cases where H-gates are mixed with other gate operations, the overall trend in execution time remains unaffected.

Based on the above observations, we propose a chip selection algorithm. We begin by filtering out chips that do not satisfy the logical qubit mapping requirements. Since the execution time scales roughly linearly with circuit depth, the relative ordering of chip performance remains stable regardless of the actual depth. This allows us to reduce the evaluation cost by using shallow circuits with the same gate rate. Specifically, we generate sample circuits with a depth of 10, matching the original circuit in both gate type ratios and qubit count. We then compile these samples on candidate chips and select the chip with the lowest Circuit Execution Cost as the preferred one.



(a) circuits with CNOT gates

(b) circuits with CNOT and H gates

(c) circuits with T gates

(d) circuits with CNOT and T gates

(e) circuits with H gates

(f) circuits with T and H gates

Fig. 7. Impact of circuit depth and gate density on execution time across different chip configurations.

While this approach reduces the cost per sampling, the overhead may still be high when the number of available chips is large. To further improve efficiency, we introduce an analytical chip preselection method. Based on circuit characteristics, we first estimate the preferred chip aspect ratio: for circuits with more T gates, chips with longer perimeters may be more suitable due to higher demand for magic state communication, whereas CNOT-heavy circuits may benefit from more square-shaped chips that reduce average communication distances. Next, leveraging our resource saturation analysis (detailed later in Section 4.3), we derive an upper bound on the chip size beyond which additional resources yield diminishing returns. Within this bounded region, we apply binary or ternary search to efficiently locate a cost-effective chip.

4.3 Quantum Chip Analyzing

Once the chip is selected, it is time to profile it. We define the *Chip Communication Capacity* C to characterize the number of parallel gates supported by a chip, denoted as C. We provide proof showing that the Chip Communication Capacityis at least $\lfloor \frac{b-1}{2} \rfloor + 3$ for a chip with bandwidth b.

Definition 4. Chip Communication Capacity:

Given a quantum chip, C is the max number u that for any u independent gates with an arbitrary placement of tiles, there exists a simultaneous execution schema for all gates.

Proposition 1. *For a chip with bandwidth b, C is always less than $4b + 2$.*

Proof:

We prove this result by contradiction. Consider a square chip with side length $(2k+2)(b+1)+b$ tiles, among which $(2k+2)(2k+2)$ are designated data tiles for logical qubits, as illustrated in Fig.8. Each channel between adjacent tiles has a communication bandwidth of b.

Now, construct a set of $4b+2$ independent CNOT gates, specifically: $2b+1$ gates between tile pairs (A_i, B_i), for $i = 1, \ldots, 2b+1$, and $2b+1$ gates between tile pairs (C_i, D_i), for $i = 1, \ldots, 2b+1$.

We categorize the chip's communication channels into internal channels (plain white) and peripheral channels (shown with gray diagonal lines). Suppose the horizontal CNOTs (A_i, B_i) and vertical CNOTs (C_i, D_i) are all routed through internal channels. In that case, their paths will inevitably intersect, causing conflicts.

To avoid this, some gates must be rerouted via peripheral channels. However, the grid's left and right channels together provide only $2b$ peripheral horizontal paths (similarly, top and bottom for vertical paths), due to the bandwidth constraint. As there are $2b+1$ vertical (or horizontal) CNOTs to route, at least one of them must still traverse an internal channel, resulting in a conflict with a gate routed in the other direction.

Therefore, it is impossible to execute $4b+2$ independent CNOT gates without exceeding the available communication bandwidth. This contradicts our assumption, and we conclude that $C < 4b+2$.

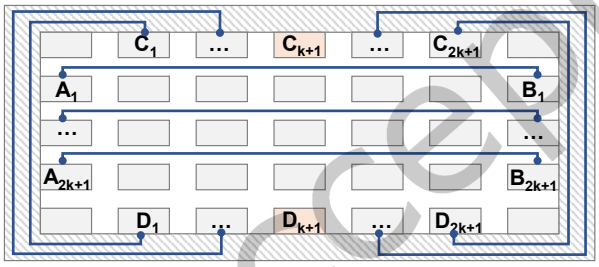
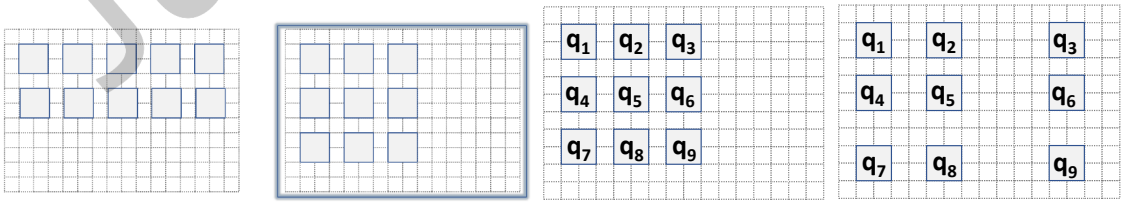


Fig. 8. A counterexample of Chip Communication Capacity

Theorem 1. *For a chip with bandwidth b, any $\lfloor \frac{b-1}{2} \rfloor + 3$ gates can be executed within the same cycle.*

Proof: As stated in Theorem 2 of previous work [49], for a chip with bandwidth b, a simultaneous path schedule exists for $\lfloor \frac{b-1}{2} \rfloor + 3$ CNOT gates, regardless of the operand qubit placement. This result can extend to a universal gate set, as T and H gates consume communication resources equivalent to CNOT gates. Magic states are generated at the chip boundary, and the chip's perimeter limits their delivery to the central region per cycle. However, this constraint is easily addressed, as the perimeter is necessarily larger than C, according to Proposition 1.



(a) (b) (c)

Fig. 9. Tile location mapping process: (a) shape determining, (b) mapping establishing, and (c) bandwidth adjusting.

4.4 Resource Sufficiency Determination

To achieve efficient transformation, we discuss our strategies under two conditions: sufficient and limited resources. We define sufficient resources as a scenario where circuit execution proceeds without delays due to path conflicts. Otherwise, it is considered a case of limited resources.

In chip characterization, our capacity calculations rely on the chip's minimum bandwidth b. Thus, when given a quantum circuit P with n qubits and a chip $L_{m_1 \times m_2}$, the first task is determining the chip's maximum minimum bandwidth b under these conditions.

$$b = \max_{t_c, t_r} \min \left\{ \left\lfloor \frac{m_1 - t_c \times t_l}{(t_c + 1) \times p_{wd}} \right\rfloor, \left\lfloor \frac{m_2 - t_r \times t_l}{(t_r + 1) \times p_{wd}} \right\rfloor \right\} \text{ subject to } t_c \times t_r \geq n.$$

Here, t_c and t_r represent the column and row numbers of the logical tile array, respectively, t_l is the side lengths of each tile, and p_{wd} is the width of the path. We can confirm that the situation can be classified as 'sufficient resources' when this lower bound $\lfloor \frac{b-1}{2} \rfloor + 3$ exceeds the circuit's $\mathbb{PM}$. Otherwise, we classified it as 'limited resources'.

4.5 Initialization

4.5.1 Location Initialization for Sufficient Resources. In this case, the location initialization strategy is straightforward: allocate logical tiles and communication resources based on the configuration that maximizes chip bandwidth (as shown in section.4.4). The logical qubits can then be mapped to the logical tiles arbitrarily.

4.5.2 Location Initialization for Limited Resources. When resources are limited, we employ the following three steps to generate a preferred tile location mapping.

Shape Determining. First, we determine the shape of the logical tile array, e.g., whether to initialize it as a 3×3 array or a 2×5 array for a circuit with nine logical qubits when both schemes are available on the chip. Then, we select an array shape with greater channel bandwidth. Among the options shown in Fig. 9a, 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. 9b. For the lattice surgery model, the communication cost is $f = \sum_{i,j} \gamma_{i,j} \times l_{i,j} + \sum_i \zeta_i \times \kappa_i$, where $l_{i,j}$ represents the Manhattan distance between the two tiles T_i, and T_j, $\gamma_{i,j}$ is the number of $CNOT_{i,j}$ in the circuit and ζ_i is the number of T gate to be executed on tile T_i, and κ_i is the minimum distance from tile T_i to the chip boundary. Intuitively, mapping qubits that frequently communicate together can reduce the communication cost. Here, we employ the Metis [26] method, an iterative graph partitioner, to generate mappings based on the qubit communication graph G_C and tile array. The communication graph G_C is used to capture the communication requirements between logical qubits, where vertices represent qubits and edges denote CNOT operations. Edge weights reflect the frequency of CNOT gates between qubit pairs. Due to the stochastic steps in the mapping generation, we generate multiple mappings and select the one with minimal communication cost.

Bandwidth Adjusting. Finally, we assign the rest of the qubit resources to each channel based on their occupancy status, as illustrated in Fig. 9c. We pre-execute each gate in the circuit to record its shortest path without considering the non-intersecting restrictions. Then, we increase the bandwidth of busy channels.

4.6 Scheduling for Lattice Surgery Model

Similar to the initialization process, we discuss the scheduling algorithm based on two types of resource availability. In the 'sufficient resources' case, the lattice surgery model can achieve an optimal solution with the method outlined in proof of Theorem 2 in Ecmas[49] to determine the corresponding paths for these gates becomes feasible.

Algorithm 2: Lattice Surgery Scheduling for Limited Resources

Input: Quantum Circuit P and Chip $L_{m_1 \times m_2}$;
Output: Encoded Circuit P^S, M_location;

```
1  tile_array = Tile_shaping (L_{m1×m2});
2  Mappings = Metis (G_C);
3  M_location = Select (Mappings, cost function);
4  while G_P not empty do
5      gates = G_P.ready_gate;
6      gates_pri = priority (gates);
7      H_weight(chip.now);
8      for g_i ← gates_pri.begin() to gates_pri.end() do
9          if g_i is CNOT gate then
10             if path (gate, chip_now) then
11                 P^S.add(shortest(path ));
12                 chip_now.update;
13         if g_i is H gate then
14             H.change_weight(chip_now);
15             if path (gate, chip_now) then
16                 P^S.add(least congested(path));
17                 chip_now.update;
18         if g_i is T gate then
19             for magic_state in exist magic state factory of chip_now do
20                 chip_now.add(virtual,magic_state);
21             if path (gate, chip_now) then
22                 P^S.add(least congested (path ));
23                 chip_now.update;
24 return P^S, M_location;
```

4.6.1 Lattice Surgery Scheduling for Limited Resources. When the resources of physical qubits are limited, finding non-intersecting paths of all gates per cycle is difficult. In such cases, we need a method to assign priorities and determine the execution order for these gates. We assign priorities to these gates to reduce latency at the bottleneck. The priority of a gate is determined by the number of remaining gates (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 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 execution process varies depending on the operation.
CNOT gate: Executing the CNOT gate is to find a path between the two tiles. If no path exists, the gate waits until the next cycle. If multiple paths exist, one is randomly selected from the shortest available paths (Line 11 in Algorithm. 2).
T gate: The implementation of a T gate can be decomposed into two steps. First, generate magic states in the magic state factory. Second, a CNOT gate will be executed between the factory and the tile where the T gate should applied. In this paper, we assume that magic states can be obtained from the boundaries of the chip when

a T gate operation is to be performed. Unlike the CNOT gate, which requires a path between two specific tiles, all magic state factories are equal for the T gate. Thus, establishing a path between the specific tile and any factories can execute the T gate (line 22). Therefore, during the execution of a T gate, we construct a virtual magic state node and establish a pathway between this virtual factory and the specific tile as shown in Fig.10a.
H gate: The H gate requires occupying three adjacent tiles within its vicinity, whereas other gates have multiple possible paths for execution. Given this, at the beginning of each execution cycle, when the set of ready gates includes H gates, we enhance the weights of the eight vicinity tiles (line 14) as shown in Fig.10b. This strategy reduces the likelihood of these tiles being selected for executing other gates.

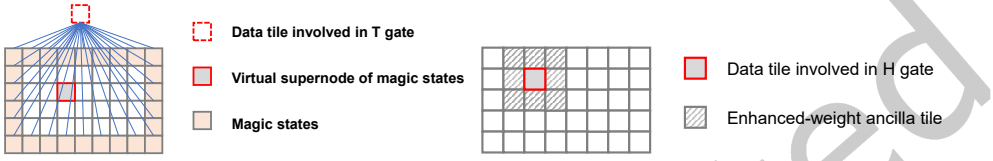


Fig. 10. Schematic of T and H Gate Execution: (a)routing of T gate using a virtual supernode; (b)resource-aware scheduling for H gate.

The time complexity of the algorithm is $O(G_N \cdot m_1 \cdot m_2)$, where G_N is the number of gates in the quantum circuit, and $m \cdot n$ is the total number of physical qubits on the chip. For each gate, the algorithm requires at most $O(m_1 \cdot m_2)$ time: for T or CNOT gates, it performs a path search over up to $O(m_1 \cdot m_2)$ candidate nodes; for H gates, the overhead is lower, involving a check over four possible options.

4.7 Optimizations for Double Defect Model

Previous works, such as Braidflash [23] and Autobraid [21], do not consider the cut type by assuming all tiles have the same cut type. However, cut type is critical in transforming the 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.

4.7.1 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. We construct a qubit communication graph where each node represents a logical qubit in the circuit, and an edge exists between two nodes if a CNOT gate is applied between the corresponding qubits. If the qubit communication graph is bipartite, we assign the same cut type to the logical qubits in one set. This is the optimal cut type initialization, with which all CNOT gates can be implemented in one cycle.

For circuits whose qubit communication graph is not bipartite, we propose a greedy algorithm that satisfies the requirement for a cut type for gates executed earlier. We construct a sub-graph of the qubit communication graph, adding the gates with no precursor in the current DAG into the sub-graph and removing these gates in the DAG. We repeat these two steps until the sub-graph is no longer bipartite with the newly added edges. The logical qubits belonging to the same set in this bipartite sub-graph are initialized to the same cut type.

4.7.2 Double Defect Scheduling for Sufficient Resources. When physical qubit resources are sufficient, we adopt the methods in Section 4.6 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 as shown in Algorithm. 3 with 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 bipartite 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 until all the gates have been scheduled. We provide the cut type scheduling algorithm with a 5/2-approximation guarantee [49].

Algorithm 3: Double Defect Scheduling for Sufficient Resources

Input: Execution Scheme $\mathcal{E}$;
Output: Encoded Circuit P^S, initialization;

```
now_step = 0;
initialization = None;
while i < E.length() do
    while G is bipartite graph do
        for gate ← E[i].begin() to E[i].end() do
            G.add_edge (gate);
        i++;
        M_c = bipartite (G);
    if initialization ≠ None then
        P^S.add (change mapping to M_c);
    else
        initialization = M_c;
    for j ← now_step to i do
        find braiding_paths (E[j]);
        P^S.add (E[j]);
    now_step = i;
return P^S, initialization;
```

4.7.3 Double Defect Scheduling for Limited Resources. When two tiles involved in a CNOT gate are of the same cut type, we estimate the impact of modifying the cut type by calculating the M-value of each tile, specifically M-value = $M_t + \theta \times M_s$. M_t denotes the impact on execution time. It takes three cycles to execute the operation directly and four cycles with the cut type modification. If this tile was idle previously, the modification operation can be performed earlier to reduce the time cost. M_s denotes the impact of the channel occupation. 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 successor gates. The parameter θ is used to determine the weights of the two factors M_t and M_s in the current situation, where $\theta = (g_r \times 2)/(b \times n)$, where g_r is the number of ready gates and b is the bandwidth of the chip. We choose to modify the type of the tile when the M-value is greater than zero.

5 Performance Evaluation

In this section, first, we compare the performance of **Ecmas+** with several state-of-the-art methods, Autobraid [21] for the double defect model and EDPCI [4] for the lattice surgery model. Then, we present the compilation

results of different circuits on various chips, demonstrating the effectiveness of our chip selection algorithm. Finally, we evaluate the scalability of our algorithm, and we highlight our key findings as follows:

- **Ecmas+** outperforms Autobraid [21], reducing the cycle of the transformed circuit by 66% at most, on average 46%, for the double defect model.
- For the lattice surgery model, **Ecmas+** reaches the optimal solution in most of the test benchmarks, reducing the cycle of the transformed circuit by 32.4% at most and on average 29.7% compared with EDPCI [4].
- **Ecmas+** can identify the most suitable chip for most benchmark circuits, achieving up to a 10.2% reduction in Circuit Execution Cost.
- **Ecmas+** exhibits excellent scalability, effectively reducing the execution time of circuits as the chip size increases while maintaining linear growth in compilation time.

5.1 Evaluation Setting

Metrics. 1) We use the number of cycles Δ to represent the execution time, serving as a metric for the effectiveness of the compilation results. 2) We define the quantum circuit's space-time cost, Circuit Execution Cost, as $\Delta \times T_n$, which reflects the usage of logical tiles and communication resources. Here, T_n is the number of tiles on the chip.
Chip Configuration. We use the same assumption proposed by EDPCI [4] that magic state factories are sufficient at the chip boundaries, where magic states can be obtained at any time. We evaluate the performance on the minimum viable chips $L_{l\times l}$, which is the smallest square grid chip that provides enough qubit with bandwidth $b = 1$.
Baselines. For the double defect and the lattice surgery models, we select the state-of-the-art algorithms Autobraid [21] and EDPCI [4] as our baselines, respectively. It should be noted that the Autobraid does not involve obtaining magic states from the boundary. Thus, in this experiment, Autobraid adopts the settings in their work where the magic state factory is inside the tile, enabling the execution of T gates to require only local CNOT gates within the tile.
Benchmarks. We use the quantum circuit from the previous works, including IBM Qiskit [36], ScaffCC [25], QUEKO [43], QASMbench [30] and random circuits. Since surface code supports a restricted set of quantum gates, we transform the circuits into circuits with only Pauli, H, T, and CNOT gates by Solovay-Kitaev decomposition in Qiskit.
Evaluation Platform. Our experiments are conducted on a system equipped with Intel®Xeon®Platinum 8360Y CPUs (36C72T per CPU, 2.4GHz), with 1TB ECC DDR4-3200MHz memory. The operating system is Ubuntu 18.04.1 LTS.

5.2 Main Result

Double Defect Model. We evaluate the performance of **Ecmas+** with minimum viable chip. As illustrated in Table 1, **Ecmas+** demonstrates superior performance over the Autobraid methods, achieving a maximum reduction of 66% in cycles, with an average reduction of 46%. In the sufficient version of our framework (**Ecmas+**-ReSu), a further reduction in cycles is observed in 27% of the circuits. Relative to Autobraid, this version realizes an average reduction of 45.8% in cycles. What stands out in this table is the remarkable optimization achieved for circuits such as QFT and BV, which can reduce execution time by up to 60%. This significant improvement is primarily due to the high density of CNOT gates in these circuits, which constitute the main component of execution time. The initialization of cut types and cut type scheduling method greatly reduces the execution time of CNOT gates, thereby decreasing the overall execution time of the circuit. **Ecmas+**-ReSu is not always the

[0]In our previous work, Ecmas [49], we evaluated only CNOT gates and removed other operations during preprocessing. In **Ecmas+**, we retain all gates (including H and T), resulting in different gate counts (g_t, g_h) and deeper circuits (α_s). This allows for more realistic and comprehensive scheduling evaluation.

Table 1. Overview of experiment result on Double Defect Model and Lattice Surgery Model

Circuit name	n	α_s^1	g_h	g_t	g_{cnot}	Autobraid	**Ecmas+**dd	**Ecmas+**dd-Re	EDPCI	**Ecmas+**ls
dnn_n8	8	541	644	352	192	739	**544**	**541**	714	**548**
grover	9	555	134	65	132	995	**583**	597	777	**555**
qpe_n9	9	154	40	18	43	318	**162**	175	225	**154**
BV_10	10	16	0	11	5	36	**16**	16	21	**16**
QFT_10	10	266	51	10	105	638	**266**	290	384	**266**
adder_n10	10	208	56	16	65	424	**228**	235	294	**208**
ising_n10	10	192	312	94	90	282	**194**	**192**	267	**194**
sat_n11	11	833	294	67	252	1649	**933**	968	1197	**833**
square_root.n4	11	855	294	62	294	1739	**981**	1011	1227	**855**
quantum_walk	11	60018	20659	3542	14372	116374	**66306**	66558	87489	**60018**
shor	12	50735	16146	4100	13838	104339	**59803**	**57596**	73047	**50735**
multiplier_n15	15	484	252	34	222	1004	**578**	586	705	**486**
qf21_n15	15	376	85	32	115	820	**388**	415	552	**376**
dnn_n16	16	541	1288	704	384	756	**596**	**541**	714	**587**
square_root_n18	18	2579	910	194	898	5151	**2915**	3026	3690	**2579**
ghz_state_n23	23	47	0	1	22	135	**47**	47	69	**47**
multiplier_n25	25	1376	770	88	670	2883	**1646**	1667	2007	**1382**
swap_test_n25	25	210	170	65	96	455	**257**	**246**	309	**210**
wstate_n27	27	57	0	2	52	168	**57**	57	84	**57**
BV_50	50	60	0	55	27	168	**60**	60	87	**60**
QFT_50	50	5126	291	50	2435	14498	**5126**	5267	7614	**5126**
ising_n50	50	39	175	66	98	57	**41**	**39**	57	**39**

[1] α_s is the depth of the circuit while g_h, g_t, and g_{cnot} represent the counts of the circuit's H, T, and CNOT gates, respectively.

best among these results. This is because, to ensure performance guarantees and avoid worst-case outcomes, the **Ecmas+**-ReSu algorithm modifies the cut types of all tiles concurrently. In the case of many single-qubit gates, each tile has more available time slots. A more greedy cut-type modification may yield better results.

Lattice Surgery Model. Ecmas+ achieves a significant reduction in cycle count, outperforming EDPCI by 29.69% execution time and providing a maximum reduction of 32.4%. For the majority of circuits, our approach obtains the optimal solutions as shown in Table 1. Given that the **Ecmas+**-ReSu for the lattice surgery model is assured to provide the optimal solution as detailed in Section 4.6, it does not require further performance evaluation in this study. Since the execution time is always greater than the depth of the logical circuit, the case where execution time equals the depth cycle count is guaranteed to be an optimal solution. We highlight this case in bold and italicized font in the table. Our method outperforms EDPCI mainly because our scheduling algorithm operates at the cycle level, while EDPCI schedules at the logic operation level. Our algorithm accounts for the varying execution times of different operations, allowing further reductions in the overall circuit execution time.

Chip Selecting. We compiled the benchmark circuits across multiple chips on the lattice surgery model. As highlighted in Table.2, our chip selection algorithm outperforms the alternatives in 13 out of 16 benchmarks, achieving the lowest Circuit Execution Cost. Our approach reduces Circuit Execution Cost by up to 10.2% compared to the baseline algorithm. The baseline algorithms we compare against are based on greedy strategies: Algorithm QF refers to the qubit resources first algorithm, which selects the smallest chip that satisfies the resource requirements, minimizing chip size. Algorithm TF selects the largest chip to minimize circuit execution time. The chip set we selected includes $L_{12t_l \times 12t_l}$, $L_{13t_l \times 13t_l}$, $L_{9t_l \times 16t_l}$, $L_{9t_l \times 17t_l}$, $L_{7t_l \times 20t_l}$, $L_{7t_l \times 21t_l}$, $L_{5t_l \times 28t_l}$, and $L_{5t_l \times 29t_l}$, varies in both size and aspect ratio, where t_l is the side length of tile. Since the experimental results for the double

Table 2. Comparison on Chip Selecting algorithm

Circuit name	n	α_s	r_t	r_h	r_{cnot}	QF_Δ	QF_{cec}	TF_Δ	TF_{cec}	**Ecmas+$_\Delta$**	**Ecmas+$_{cec}$**
dnn_n8	8	541	0.338	0.185	0.101	596	83440	550	92950	586	**82040**
grover	9	555	0.057	0.028	0.057	555	77700	557	94133	557	77980
qpe_n9	9	154	0.059	0.027	0.064	154	21560	154	26026	154	**21560**
BV_10	10	16	0.0	0.157	0.071	16	2240	16	2704	16	**2240**
QFT_10	10	266	0.04	0.008	0.082	266	37240	266	44954	266	**37240**
adder_n10	10	208	0.057	0.016	0.066	210	29400	208	35152	209	**29260**
ising_n10	10	192	0.351	0.106	0.101	199	27860	192	32448	204	28560
sat_n11	11	833	0.067	0.015	0.057	834	116760	833	140777	833	**116620**
square_root.n4	11	855	0.065	0.014	0.065	862	120680	855	144495	858	**120120**
multiply_n13	13	79	0.083	0.024	0.079	88	12320	79	13351	79	**11060**
multiplier_n15	15	484	0.071	0.01	0.063	488	68320	484	81796	486	**68040**
qf21_n15	15	376	0.031	0.012	0.042	376	52640	376	63544	377	52780
dnn_n16	16	541	0.338	0.185	0.101	637	89180	602	101738	617	**86380**
ghz_state_n23	23	47	0.0	0.002	0.042	47	6580	47	7943	47	**6580**
swap_test_n25	25	210	0.066	0.025	0.037	214	29960	217	36673	213	29820
wstate_n27	27	57	0.0	0.003	0.069	57	8208	57	9633	57	**8208**

defect model are similar to those for the lattice surgery model with only CNOT gates, we have omitted the double defect experiments. Moreover, because the gates in the current circuits are relatively sparse, the differences in execution time across different chips are more pronounced for denser circuits.

5.3 Sensitivity Study

In this section, we undertake sensitivity analysis focused on location and cut type initialization, gate prioritization, and modification of cut types.

5.3.1 Location Mapping and Gate Scheduling: As illustrated in Table 3, our method consistently shows superior performance in most circuits. Compared to the Basis algorithm, our algorithm achieves an average improvement of 11% in circuit execution time and a maximum reduction of 53% in circuit execution time. The **bold results** in the table indicate the cases where the cycle count of our algorithm and circuit depth are the same, which necessarily represents the optimal solution. In other cases, due to the NP-hard nature of the problem, it is challenging to determine whether the optimal solution has been achieved under the given resource constraints. The Basis method starts with a random mapping and executes operations layer by layer. The layered approach executes one layer of quantum operations from the logical circuit at a time before proceeding to the next layer, similar to the strategy used in EDPCI [4]. The **Ecmas+$_{sch}$** method uses the same random mapping result of Basis but applies our scheduling algorithm. The **Ecmas+$_{map}$** method combines our algorithm's mapping with layered scheduling. Finally, Ours combines both our mapping and scheduling algorithms. Experimental results show that our scheduling algorithm significantly reduces circuit execution time, while our mapping algorithm is effective in circuits with a high Circuit Parallelism Degree, such as the swap_test_n25 circuit and ising_n50 circuit.

5.3.2 Cut Type Mapping and Cut Type Scheduling: According to the results in Table 4, our method consistently outperforms the baseline algorithms across all tested circuits, achieving up to a 58% reduction in execution time with an average reduction of 43% execution time compared to the Basis$_{cut}$ algorithm. The Basis$_{cut}$ algorithm

Table 3. Comparison of location initialization and gate scheduling in lattice surgery model

Circuit name	n	α_s	g_t	g_h	g_{cnot}	Basis	**Ecmas+**$_{sch}$	**Ecmas+**$_{map}$	**Ecmas+**
dnn_n8	8	541	644	352	192	573	550	572	545
grover	9	555	134	65	132	576	555	576	**555**
qpe_n9	9	154	40	18	43	160	154	160	**154**
QFT_10	10	266	51	10	105	266	266	266	**266**
adder_n10	10	208	56	16	65	214	208	212	**208**
ising_n10	10	192	312	94	90	245	193	238	194
multiply_n13	13	79	42	12	40	87	79	86	**79**
square_root_n18	18	2579	910	194	898	2616	2579	2616	**2579**
swap_test_n25	25	210	170	65	96	253	210	243	**210**
ising_n50	50	39	175	66	98	84	57	60	**39**

initializes all tiles with the same cut type and does not change the cut type during circuit execution. The **Ecmas+**$_{cut\_sch}$ approach uses the same initial cut type mapping but applies our cut type modification algorithm. The **Ecmas+**$_{cut\_map}$ approach employs our cut type mapping algorithm but does not change the cut type during scheduling. The location mapping and gate scheduling methods are consistent with **Ecmas+** in all four approaches. It is evident that applying either the mapping or scheduling algorithm alone can significantly optimize circuit execution time. For circuits with a large number of CNOT gates, the scheduling algorithm yields more pronounced improvements due to the complex communication relationships between tiles.

Table 4. Comparison of cut type initialization and modification in double defect model

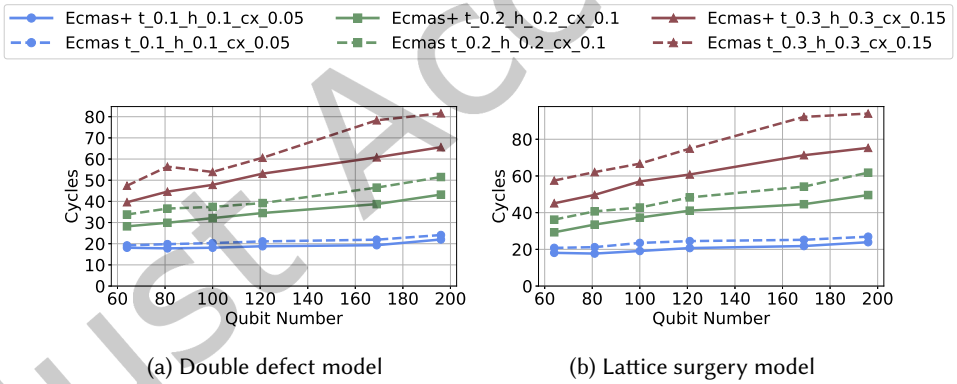
Circuit name	n	α_s	g_t	g_h	g_{cnot}	Basis$_{cut}$	**Ecmas+**$_{cut\_sch}$	**Ecmas+**$_{cut\_map}$	**Ecmas+**
dnn_n8	8	541	644	352	192	760	547	547	**547**
grover	9	555	134	65	132	995	586	803	**583**
qpe_n9	9	154	40	18	43	318	165	222	**162**
QFT_10	10	266	51	10	105	638	269	565	**266**
adder_n10	10	208	56	16	65	424	234	284	**228**
ising_n10	10	192	312	94	90	281	205	193	**193**
multiply_n13	13	79	42	12	40	171	99	127	**96**
square_root_n18	18	2579	910	194	898	5151	2918	4239	**2915**
swap_test_n25	25	210	170	65	96	463	262	304	**255**
ising_n50	50	39	175	66	98	85	46	45	**45**

5.3.3 Comparison with Ecmas. To quantitatively assess the improvements brought by **Ecmas+**, we extend the original Ecmas framework to naively support universal gate sets without introducing our enhanced mapping, and scheduling. We compare Ecmas and **Ecmas+** across both benchmark circuits and randomly generated circuits. As summarized in Table 5, under double defect model, **Ecmas+** achieves up to a 24% reduction in cycles, while under lattice surgery model, the maximum reduction reaches 17%. Although some benchmark circuits show minimal improvement, this is mainly due to the low density of operations in these circuits, which limits optimization potential. To better assess the impact of operation density, we additionally generated high-density random

circuits. As shown in Fig.11, both the number of gates and gate density positively correlate with the performance gains of **Ecmas+**, with greater improvements observed as gate number increases. Each data point represents the average of 10 random circuits, all with a depth of 10. The gate rate configuration (e.g., "t_0.1_h_0.1_cx_0.05" as a representative example) indicates that, in each layer, the number of T, H, and CX gates is set to 10%, 10%, and 5% of the total number of qubits, respectively.

Table 5. Comparison of **Ecmas**+ and Ecmas

Circuit name	n	α_s	g_t	g_h	g_{cnot}	Ecmas$_{dd}$	**Ecmas+**$_{dd}$	rate$_{dd}$	Ecmas$_{ls}$	**Ecmas+**$_{ls}$	rate$_{ls}$
dnn_n8	8	541	644	352	192	563	546	0.03	575	557	0.03
grover	9	555	134	65	132	583	587	-0.01	556	556	0.00
qpe_n9	9	154	40	18	43	162	162	0.00	154	154	0.00
QFT_10	10	266	51	10	105	266	266	0.00	266	266	0.00
adder_n10	10	208	56	16	65	228	228	0.00	208	208	0.00
ising_n10	10	192	312	94	90	195	192	0.02	194	193	0.01
multiply_n13	13	79	42	12	40	95	95	0.00	79	79	0.00
square_root_n18	18	2579	910	194	898	2917	2915	0.00	2585	2579	0.00
swap_test_n25	25	210	170	65	96	268	256	0.04	216	210	0.03
ising_n50	50	39	175	66	98	54	41	0.24	54	45	0.17



(a) Double defect model (b) Lattice surgery model

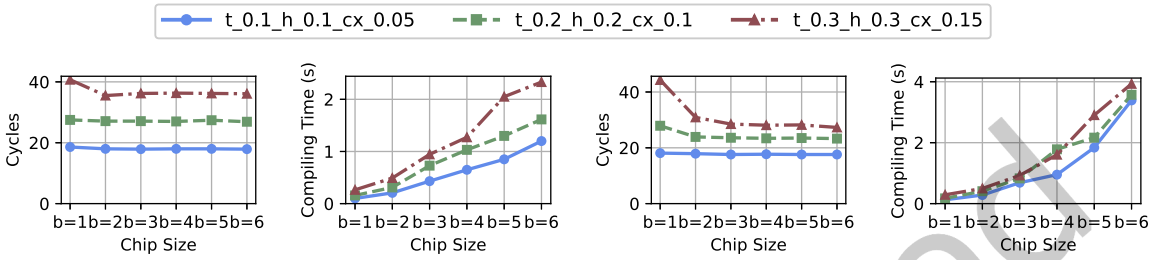
Fig. 11. Comparison of **Ecmas**+ and Ecmas on random circuits.

5.4 Scalability

We evaluate the scalability of **Ecmas+** to both the size of the quantum circuits and the quantum chips. Each data point represents the average of 10 randomly generated circuits with a depth of 10.

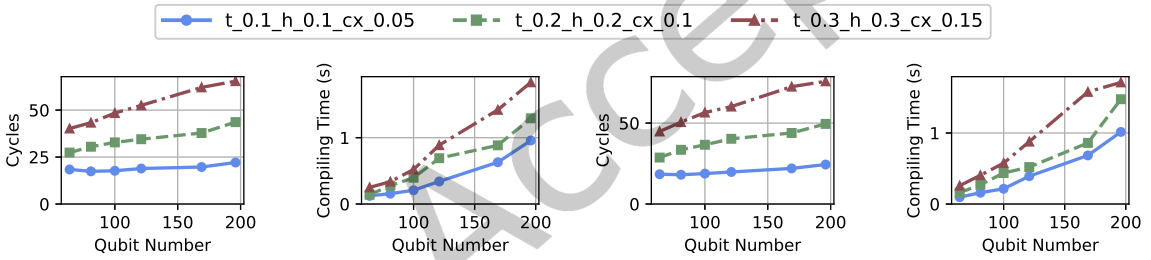
Scalability over chip size. As shown in Fig. 12, the cycle count of our method decreases as the chip size increases, indicating that the additional qubit resources are effectively utilized to enhance spatial scheduling efficiency. Meanwhile, the compilation time increases linearly with the chip size, demonstrating that our framework incurs a manageable overhead when scaling up. Each random circuit consists of 64 qubits.

Scalability over qubit number. Fig. 13 shows that the compilation time scales linearly with the qubit number, highlighting the efficiency of our design when dealing with larger circuits. Each point is the average over 10 random circuits with a fixed depth of 10. The chip size used here corresponds to the minimal viable square chip, i.e., a configuration where the average bandwidth is 1.



(a) DD: execution time (b) DD: compilation time (c) LS: execution time (d) LS: compilation time

Fig. 12. Scalability of **Ecmas+** on chipsize. DD refers to double defect model, and LS refers to lattice surgery model.



(a) DD: execution time (b) DD: compilation time (c) LS: execution time (d) LS: compilation time

Fig. 13. Scalability of **Ecmas+** on qubit number. DD refers to double defect model, and LS refers to lattice surgery model.

6 Related Work

Quantum compilation can be categorized into recent NISQ (Noisy Intermediate-Scale Quantum) compilation and long-term fault-tolerant compilation. Most existing quantum compilers are designed for physical qubit-level compilation for NISQ circuits focusing on the gap between logical circuits and hardware constraints. Extensive research [13, 14, 31, 42, 50] focuses on resolving the challenge of executing CNOT gates on superconducting chips. Some works are also dedicated to compiling photonic quantum computers [46, 47]. As qubit numbers rise, fault-tolerant quantum computation is becoming a reality. This development necessitates a new compilation framework capable of converting circuits into forms encoded with quantum error correction (QEC) codes [29, 34, 44].

Different QEC codes impose various constraints, and our work primarily focuses on the surface code, which is currently the mainstream QEC code. Depending on the method of constructing logical qubits, surface code can be divided into the double defect model [12] and the lattice surgery model [20] with different logical operation implementation strategies. Double defect employs the braiding technique to perform CNOT gates. Braidflash

[23] abstracts the constraints of CNOT gates implementation into braiding path disjoint. Autobraid [21] further discovers the local parallelism pattern, enabling efficient search for as many parallel CNOT gates as possible. However, neither approach considers the cut type issue, which contributes to the prolonged execution time of CNOT gates. Lattice surgery is a novel entrant in surface code approaches, employing fewer physical qubits for encoding a logical qubit. It utilizes ZZ measurements for CNOT gate [32]. However, the physical adjacency requirement for measurements limits its effectiveness. As a result, executing long-range CNOT gates is time-consuming. EDPCI [4] achieves long-range CNOT gates by utilizing ancilla tiles to construct Bell states and maximizes communication parallelism using disjoint pathfinding methods.

As quantum technology advances, more research institutions offer quantum cloud services using chip clusters. Compiler optimization has shifted from single-chip circuit translation to cluster-level improvements [7, 33, 37], focusing on enhancing throughput and efficiency.

7 Discussion

Generality and Applicability of Ecmas+ While our evaluation focuses on surface codes, the core principles of **Ecmas+** are agnostic to the choice of quantum error correction code. Its strength lies in dynamically allocating communication resources based on circuit demands. This makes **Ecmas+** applicable to a broad range of encoding schemes. For color codes employing lattice surgery with XX/ZZ measurements, **Ecmas+** can optimize communication ancilla placement, with chip layout adjustments to match their triangular tiling. Moreover, **Ecmas+** extends to unencoded circuits where long-range CNOTs are executed via Bell pairs or teleportation. In these settings, it assists in ancilla region placement and reuse, supporting space-time trade-offs beyond SWAP-based routing.

Current Limitations While **Ecmas+** demonstrates promising performance across a range of benchmarks, several limitations remain. First, our chip selection and initial mapping strategies are primarily heuristic and guided by empirical intuition; they currently lack theoretical guarantees or formal optimization foundations. Second, the present framework focuses solely on the scheduling of communication resources without considering other critical ancilla-consuming components, such as magic state factories.

Future Directions To further broaden the applicability of **Ecmas+**, future work can explore multi-circuit execution on a single chip. This introduces challenges such as spatial isolation, resource contention, and heterogeneous fault tolerance requirements. We also plan to adapt **Ecmas+** to modular systems like IBM's Kookaburra [1], where inter-module links must be efficiently managed. Tackling these challenges is essential for scaling quantum systems to realistic multi-programmed workloads.

8 Conclusion

In this paper, we first study the chip selection problem on the quantum cloud platform. Then, we study the surface code mapping and scheduling problems in both double defect and lattice surgery models. We introduce Circuit Parallelism Degree to quantitatively analyze quantum circuits and propose a chip selection algorithm that selects the chip with the lowest overall cost based on circuit characteristics. Next, we characterize the available chip resources to assess whether they are sufficient. Our transformer features algorithms for scenarios with limited and sufficient qubit resources. Extensive evaluations show that **Ecmas+** provides significant reduction over the SOTA approaches by reducing the execution time by 46% on average for the double defect model and 29.7% execution time for the lattice surgery model.

9 Acknowledgments

This work was supported by Innovation Program for Quantum Science and Technology (Grant No. 2021ZD0302901), Anhui Initiative in Quantum Information Technologies (Grant No. AHY150100), and National Natural Science

Foundation of China (Grant No. 62102388). Xiang-Yang Li and Wei Xie are the corresponding authors (Email: xiangyangli@ustc.edu.cn and xxieww@ustc.edu.cn).

References

[1] Muhammad AbuGhanem. 2025. IBM quantum computers: Evolution, performance, and future directions. *The Journal of Supercomputing* 81, 5 (2025), 687.

[2] Rajeev Acharya, Igor Aleiner, Richard Allen, Trond I. Andersen, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Juan Atalaya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Joao Basso, Andreas Bengtsson, Sergio Boixo, Gina Bortoli, Alexandre Bourassa, Jenna Bovaird, Leon Brill, Michael Broughton, Bob B. Buckley, David A. Buell, Tim Burger, Brian Burkett, Nicholas Bushnell, Yu Chen, Zijun Chen, Ben Chiaro, Josh Cogan, Roberto Collins, Paul Conner, William Courtney, Alexander L. Crook, Ben Curtin, Dripto M. Debroy, Alexander Del Toro Barba, Sean Demura, Andrew Dunsworth, Daniel Eppens, Catherine Erickson, Lara Faoro, Edward Farhi, Reza Fatemi, Leslie Flores Burgos, Ebrahim Forati, Austin G. Fowler, Brooks Foxen, William Giang, Craig Gidney, Dar Gilboa, Marissa Giustina, Alejandro Grajales Dau, Jonathan A. Gross, Steve Habegger, Michael C. Hamilton, Matthew P. Harrigan, Sean D. Harrington, Oscar Higgott, Jeremy Hilton, Markus Hoffmann, Sabrina Hong, Trent Huang, Ashley Huff, William J. Huggins, Lev B. Ioffe, Sergei V. Isakov, Justin Iveland, Evan Jeffrey, Zhang Jiang, Cody Jones, Pavol Juhas, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Tanuj Khattar, Mostafa Khezri, Mária Kieferová, Seon Kim, Alexei Kitaev, Paul V. Klimov, Andrey R. Klots, Alexander N. Korotkov, Fedor Kostritsa, John Mark Kreikebaum, David Landhuis, Pavel Laptev, Kim-Ming Lau, Lily Laws, Joonho Lee, Kenny Lee, Brian J. Lester, Alexander Lill, Wayne Liu, Aditya Locharla, Erik Lucero, Fionn D. Malone, Jeffrey Marshall, Orion Martin, Jarrod R. McClean, Trevor McCourt, Matt McEwen, Anthony Megrant, Bernardo Meurer Costa, Xiao Mi, Kevin C. Miao, Masoud Mohseni, Shirin Montazeri, Alexis Morvan, Emily Mount, Wojciech Mruczkiewicz, Ofer Naaman, Matthew Neeley, Charles Neill, Ani Nersisyan, Hartmut Neven, Michael Newman, Jiun How Ng, Anthony Nguyen, Murray Nguyen, Murphy Yuezhen Niu, Thomas E. O'Brien, Alex Opremcak, John Platt, Andre Petukhov, Rebecca Potter, Leonid P. Pryadko, Chris Quintana, Pedram Roushan, Nicholas C. Rubin, Negar Saei, Daniel Sank, Kannan Sankaragomathi, Kevin J. Satzinger, Henry F. Schurkus, Christopher Schuster, Michael J. Shearn, Aaron Shorter, Vladimir Shvarts, Jindra Skruzny, Vadim Smelyanskiy, W. Clarke Smith, George Sterling, Doug Strain, Marco Szalay, Alfredo Torres, Guifre Vidal, Benjamin Villalonga, Catherine Vollgraff Heidweiller, Theodore White, Cheng Xing, Z. Jamie Yao, Ping Yeh, Juhwan Yoo, Grayson Young, Adam Zalcman, Yaxing Zhang, Ningfeng Zhu, and Google Quantum AI. 2023. Suppressing quantum errors by scaling a surface code logical qubit. 614, 7949 (2023), 676–681. doi:10.1038/s41586-022-05434-1

[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.

[4] Michael Beverland, Vadym Kliuchnikov, and Eddie Schoute. 2022. Surface code compilation via edge-disjoint paths. *PRX Quantum* 3, 2 (2022), 020342. https://github.com/eddieschoute/TeleportRouter.jl

[5] Sergey B Bravyi and A Yu Kitaev. 1998. Quantum codes on a lattice with boundary. *arXiv preprint quant-ph/9811052* (1998).

[6] Weizhou Cai, Xianghao Mu, Weiting Wang, Jie Zhou, Ma Yuwei, Xiaoxuan Pan, Ziyue Hua, Xinyu Liu, Guangming Xue, Haifeng Yu, Haiyan Wang, Yipu Song, and Chang-Ling Zou. 2024. Protecting entanglement between logical qubits via quantum error correction. *Nature Physics* 20 (03 2024), 1–5. doi:10.1038/s41567-024-02446-8

[7] Poulami Das, Swamit S. Tannu, Prashant J. Nair, and Moinuddin Qureshi. 2019. A Case for Multi-Programming Quantum Computers. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO-52)*. Association for Computing Machinery, New York, NY, USA, 291–303. doi:10.1145/3352460.3358287

[8] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. 2002. Topological quantum memory. *J. Math. Phys.* 43, 9 (2002), 4452–4505.

[9] Cirq Developers. 2024. *Cirq.* doi:10.5281/zenodo.11398048

[10] Yongshan Ding, Adam Holmes, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic T. Chong. 2018. Magic-state functional units: mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture* (Fukuoka, Japan) *(MICRO-51)*. IEEE Press, 828–840. doi:10.1109/MICRO.2018.00072

[11] Gerd Finke, Pierre Lemaire, Jean-Marie Proth, and Maurice Queyranne. 2009. Minimizing the number of machines for minimum length schedules. *European Journal of Operational Research* 199, 3 (2009), 702–705.

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

[13] Hao Fu, Mingzheng Zhu, Fangzheng Chen, Chi Zhang, Jun Wu, Wei Xie, and Xiang-Yang Li. 2024. Effective and Efficient Parallel Qubit Mapper. *Trans. Comp.-Aided Des. Integ. Cir. Sys.* 44, 5 (Nov. 2024), 1774–1787. doi:10.1109/TCAD.2024.3500784

[14] Hao Fu, Mingzheng Zhu, Jun Wu, Wei Xie, Zhaofeng Su, and Xiang-Yang Li. 2023. Effective and Efficient Qubit Mapper. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD57390.2023.10323857

[15] Iulia M Georgescu, Sahel Ashhab, and Franco Nori. 2014. Quantum simulation. *Reviews of Modern Physics* 86, 1 (2014), 153.
[16] Craig Gidney and Martin Ekerå. 2021. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum* 5 (2021), 433.
[17] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96)*. Association for Computing Machinery, New York, NY, USA, 212–219. doi:10.1145/237814.237866
[18] Vojtěch Havlíček, Antonio D Córcoles, Kristan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gambetta. 2019. Supervised learning with quantum-enhanced feature spaces. *Nature* 567, 7747 (2019), 209–212.
[19] Daniel Herr, Franco Nori, and Simon J Devitt. 2017. Optimization of lattice surgery is NP-hard. *Npj quantum information* 3, 1 (2017), 35.
[20] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. 2012. Surface code quantum computing by lattice surgery. *New Journal of Physics* 14, 12 (2012), 123011.
[21] Fei Hua, Yanhao Chen, Yuwei Jin, Chi Zhang, Ari Hayes, Youtao Zhang, and Eddy Z. Zhang. 2021. AutoBraid: A Framework for Enabling Efficient Surface Code Communication in Quantum Computing. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 925–936. doi:10.1145/3466752.3480072
[22] Hsin-Yuan Huang, Michael Broughton, Masoud Mohseni, Ryan Babbush, Sergio Boixo, Hartmut Neven, and Jarrod R McClean. 2021. Power of data in quantum machine learning. *Nature communications* 12, 1 (2021), 1–9.
[23] Ali Javadi-Abhari, Pranav Gokhale, Adam Holmes, Diana Franklin, Kenneth R. Brown, Margaret Martonosi, and Frederic T. Chong. 2017. Optimized surface code communication in superconducting quantum computers. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (Cambridge, Massachusetts) *(MICRO-50 '17)*. Association for Computing Machinery, New York, NY, USA, 692–705. doi:10.1145/3123939.3123949
[24] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. 2024. Quantum computing with Qiskit. arXiv:2405.08810 [quant-ph] doi:10.48550/arXiv.2405.08810
[25] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. 2014. ScaffCC: a framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers* (Cagliari, Italy) *(CF '14)*. Association for Computing Machinery, New York, NY, USA, Article 1, 10 pages. doi:10.1145/2597917.2597939
[26] George Karypis. 1997. METIS: Unstructured graph partitioning and sparse matrix ordering system. *Technical report* (1997).
[27] A Yu Kitaev. 2003. Fault-tolerant quantum computation by anyons. *Annals of physics* 303, 1 (2003), 2–30.
[28] Sebastian Krinner, Nathan Lacroix, Ants Remm, Agustin Di Paolo, Elie Genois, Catherine Leroux, Christoph Hellings, Stefania Lazar, Francois Swiadek, Johannes Herrmann, et al. 2022. Realizing repeated quantum error correction in a distance-three surface code. *Nature* 605, 7911 (2022), 669–674.
[29] Lingling Lao, Bas van Wee, Imran Ashraf, J van Someren, Nader Khammassi, Koen Bertels, and Carmen G Almudever. 2018. Mapping of lattice surgery-based quantum circuits on surface code architectures. *Quantum Science and Technology* 4, 1 (2018), 015005.
[30] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. 2023. QASMBench: A Low-Level Quantum Benchmark Suite for NISQ Evaluation and Simulation. *ACM Transactions on Quantum Computing* 4, 2, Article 10 (Feb. 2023), 26 pages. doi:10.1145/3550488
[31] Gushu Li, Yufei Ding, and Yuan Xie. 2019. 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* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 1001–1014. doi:10.1145/3297858.3304023
[32] Daniel Litinski. 2019. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum* 3 (2019), 128.
[33] Lei Liu and Xinglei Dou. 2021. QuCloud: A New Qubit Mapping Mechanism for Multi-programming Quantum Computing in Cloud Environment. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 167–178. doi:10.1109/HPCA51647.2021.00024
[34] Daniel Nigg, Markus Mueller, Esteban A Martinez, Philipp Schindler, Markus Hennrich, Thomas Monz, Miguel A Martin-Delgado, and Rainer Blatt. 2014. Quantum computations on a topologically encoded qubit. *Science* 345, 6194 (2014), 302–305.
[35] Joe O'Gorman and Earl T. Campbell. 2017. Quantum computation with realistic magic-state factories. *Phys. Rev. A* 95 (Mar 2017), 032338. Issue 3. doi:10.1103/PhysRevA.95.032338
[36] Qiskit contributors. 2023. Qiskit: An Open-source Framework for Quantum Computing. doi:10.5281/zenodo.2573505
[37] Gokul Subramanian Ravi, Kaitlin N. Smith, Pranav Gokhale, and Frederic T. Chong. 2021. Quantum Computing in the Cloud: Analyzing job and machine characteristics . In *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE Computer Society, Los Alamitos, CA, USA, 39–50. doi:10.1109/IISWC53511.2021.00015
[38] Markus Reiher, Nathan Wiebe, Krysta M Svore, Dave Wecker, and Matthias Troyer. 2017. Elucidating reaction mechanisms on quantum computers. *Proceedings of the national academy of sciences* 114, 29 (2017), 7555–7560.

[39] Maria Schuld and Nathan Killoran. 2019. Quantum machine learning in feature Hilbert spaces. *Physical review letters* 122, 4 (2019), 040504.
[40] P.W. Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 124–134. doi:10.1109/SFCS.1994.365700
[41] Peter W Shor. 1996. Fault-tolerant quantum computation. In *Proceedings of 37th Conference on Foundations of Computer Science*. IEEE, 56–65.
[42] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintão Pereira. 2018. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. 113–125.
[43] Bochen Tan and Jason Cong. 2020. Optimality study of existing quantum computing layout synthesis tools. *IEEE Trans. Comput.* 70, 9 (2020), 1363–1373.
[44] Christophe Vuillot, Lingling Lao, Ben Criger, Carmen García Almudéver, Koen Bertels, and Barbara M Terhal. 2019. Code deformation and lattice surgery are gauge fixing. *New Journal of Physics* 21, 3 (2019), 033028.
[45] Yulin Wu, Wan-Su Bao, Sirui Cao, Fusheng Chen, Ming-Cheng Chen, Xiawei Chen, Tung-Hsun Chung, Hui Deng, Yajie Du, Daojin Fan, et al. 2021. Strong quantum computational advantage using a superconducting quantum processor. *Physical review letters* 127, 18 (2021), 180501.
[46] Hezi Zhang, Jixuan Ruan, Hassan Shapourian, Ramana Rao Kompella, and Yufei Ding. 2024. OnePerc: A Randomness-aware Compiler for Photonic Quantum Computing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 738–754.
[47] Hezi Zhang, Anbang Wu, Yuke Wang, Gushu Li, Hassan Shapourian, Alireza Shabani, and Yufei Ding. 2023. OneQ: A Compilation Framework for Photonic One-Way Quantum Computation. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 12, 14 pages. doi:10.1145/3579371.3589047
[48] Youwei Zhao, Yangsen Ye, He-Liang Huang, Yiming Zhang, Dachao Wu, Huijie Guan, Qingling Zhu, Zuolin Wei, Tan He, Sirui Cao, et al. 2022. Realization of an error-correcting surface code with superconducting qubits. *Physical Review Letters* 129, 3 (2022), 030501.
[49] Mingzheng Zhu, Hao Fu, Jun Wu, Chi Zhang, Wei Xie, and Xiang-Yang Li. 2024. Ecmas: Efficient Circuit Mapping and Scheduling for Surface Code. In *2024 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 158–169.
[50] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2018. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 7 (2018), 1226–1236.

Received 20 October 2024; revised 8 May 2025; accepted 5 August 2025