CLASS: A Controller-Centric Layout Synthesizer for Dynamic Quantum Circuits

Yu Chen^{1,2,†}, Yilun Zhao^{1,2,†}, Bing Li^{3}, He Li^{4}, Mengdi Wang^{1}, Yinhe Han^{1}, and Ying Wang^{1,*}
Research Center for Intelligent Computing Systems, Institute of Computing Technology, Chinese Academy of Sciences[1]
University of Chinese Academy of Sciences[2], *Institute of Microelectronics, Chinese Academy of Sciences*[3], *Southeast University*[4]
Emails: {chenyu21b,zhaoyilun22b,wangmengdi,yinhes,wangying2009}@ict.ac.cn,libing2024@ime.ac.cn,helix@seu.edu.cn

***Abstract*—Layout Synthesis for Quantum Computing (LSQC) is a critical component of quantum design tools. Traditional LSQC studies primarily focus on optimizing for reduced circuit depth by adopting a device-centric design methodology. However, these approaches overlook the impact of classical processing and communication time, thereby being insufficient for Dynamic Quantum Circuits (DQC).**

To address this, we introduce CLASS, a controller-centric layout synthesizer designed to reduce inter-controller communication latency in a distributed control system. It consists of a two-stage framework featuring a hypergraph-based modeling and a heuristic-based graph partitioning algorithm. Evaluations demonstrate that CLASS effectively reduces communication latency by up to 100% with only a 2.10% average increase in the number of additional operations.

***Index Terms*—quantum computing, layout synthesis**

I. Introduction

Similar to the design of classical circuits and systems, realizing conceptual quantum algorithms on actual devices requires a multitude of complex design tasks [1]. One of the most challenging design tasks is *Qubit mapping* [2], or *Layout Synthesis for Quantum Computing* (LSQC) [3].

Prior LSQC studies have predominantly followed a *device-centric* design methodology, with a primary focus on minimizing the execution time of quantum operations on quantum devices. As a result, substantial effort has been devoted to reducing the number of SWAP gates induced by topological constraints of physical qubits. [2]–[10].

While the device-centric methodology is effective for *static* quantum circuits, it falls short in optimizing for *Dynamic* Quantum Circuits (DQC), a promising paradigm essential for various quantum experiments [11]–[15]. The primary reason is that a quantum program's execution time depends on the instruction processing time on Quantum Control Processors (QCPs) [16]. In static circuits, this processing time is comparable to the duration of quantum device operations (cf. Sec. II-D), making the device-centric approach reasonable. In contrast, DQCs often involve frequent mid-circuit measurements and feedforward operations [11], [12], introducing additional latency that cannot be captured by on-device quantum operations. Therefore, a controller-centric design methodology becomes essential to accurately account for the total execution latency on QCPs.

[*]Corresponding author.
[†]Equal contribution.

A. Controller-Centric LSQC Challenge

Achieving quantum advantage requires scaling up to thousands or even millions of qubits [17], [18]. To support such large-scale quantum systems, it is natural to adopt a distributed control architecture composed of numerous QCPs.

When executing DQCs on distributed control systems, inter-controller communication becomes a critical performance bottleneck. Specifically, feedforward operations on certain qubits may depend on measurement outcomes from qubits managed by different QCPs. As a result, inter-controller communication is required to exchange measurement results or branching flags—an expensive operation in quantum computing due to the limited coherence time of qubits. Moreover, the latency of such communication inevitably increases with system scale.

To mitigate this overhead and preserve execution fidelity, two complementary approaches can be considered. On the hardware side, latency can be reduced through improved communication protocols and specialized interconnects. On the software side, the mapping of logical qubits to controllers—determined by the layout synthesizer—plays a crucial role. For instance, if all qubits involved in a feedforward operation are assigned to the same controller, the operation can be executed locally, eliminating inter-controller communication entirely.

This leads to a clear design objective for controller-centric layout synthesis:

Design Goal

For each measurement-feedforward operation, the involved qubits should be mapped to a set of controllers that minimizes inter-controller communication latency.

Unfortunately, existing LSQC solutions are designed without considering the above objective, making their modeling approaches difficult to adapt to this emerging problem. Therefore, it remains an open challenge to design a controller-centric layout synthesizer to optimize for lower inter-controller communication latency.

B. Contributions

In this paper, we present CLASS, a systematic approach to address the emerging LSQC challenge in distributed control systems. Our key contributions are as follows:

1. **A controller-centric design methodology.** Through detailed analysis, we identify the instruction processing time of quantum control systems as the dominant factor affecting program execution time. This insight reveals that considering only on-device operation latency is insufficient when designing tools for quantum computing.
2. **CLASS: a Controller-centric LAyout SyntheSizer.** CLASS reformulates the LSQC problem as a hypergraph partitioning task, enabling a concise and modular algorithmic framework that can be seamlessly integrated into existing layout synthesis pipelines.
3. **Evaluation of CLASS.** We evaluate CLASS across a variety of DQC benchmarks, demonstrating its effectiveness in reducing inter-controller communication latency. Compared to existing synthesizers, CLASS achieves an average 48.45% reduction in inter-controller communication hops, with only a 2.10% average increase in additional operations. Our implementation is open-source[1].

II. Background and Motivation

In this section, we commence with a concise overview of existing layout synthesizers. Next, we introduce the concepts of dynamic quantum circuits and quantum control systems, providing examples to aid readers in understanding the specific problem addressed in this paper. Basic concepts of quantum computing theory have been omitted for brevity; readers seeking foundational knowledge may refer to textbooks such as Ref. [19] for an in-depth introduction.

A. Layout Synthesis for Quantum Computing

A *quantum circuit*[2] is a graphical representation of a quantum algorithm, consisting of a sequence of quantum gates or operations applied to logical qubits[3]. Implementing two-qubit gates requires physical connectivity, but the limited connectivity of most quantum devices often renders quantum circuits non-executable on such hardware. To address this challenge, LSQC has become a critical component of modern quantum design tools, which typically consists of two stages: (1) generating an *initial placement*, which maps logical qubits to physical qubits, and (2) producing a *gate schedule*, which determines where and when to apply SWAP operations to enable the circuit's execution on the target device. Previous studies on LSQC can be categorized into three main approaches. The first employs heuristic search strategies, modeling the quantum circuit as a directed acyclic graph (DAG) and using breadth-first search-like methods for SWAP insertion [2], [20], [21]. The second formulates LSQC as a mathematical optimization problem and solves it using specialized solvers [3], [10], [22]–[24]. The third leverages machine learning techniques, such as reinforcement learning, to tackle LSQC [9], [25]. Among these, heuristic methods are the most widely adopted due to their efficiency and stability. For example, SABRE [2], one of the most prominent heuristic layout synthesizers, has been integrated into IBM's Qiskit framework [26], the most widely used quantum computing software. It is important to note that all the aforementioned approaches primarily focus on *static* quantum circuits without measurement feedforward operations, making them complementary to our study.

[1]https://github.com/Zhaoyilunnn/dqc-map

[2]We use quantum circuit and quantum program interchangeably throughout this paper.

[3]The term "logical" in this work does not refer to quantum error correction.

B. Dynamic Quantum Circuits

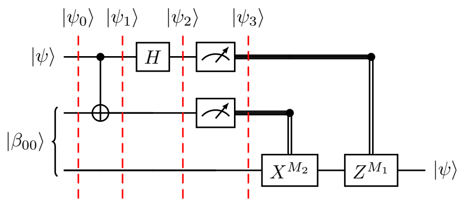


Fig. 1: Quantum circuit for teleporting a qubit.

$$\begin{aligned} |\psi_1\rangle &= \frac{1}{\sqrt{2}}\Big[\alpha|0\rangle\left(|00\rangle+|11\rangle\right)+\beta|1\rangle\left(|10\rangle+|01\rangle\right)\Big], \\ |\psi_2\rangle &= \frac{1}{2}\Big[\alpha\left(|0\rangle+|1\rangle\right)\left(|00\rangle+|11\rangle\right)+\beta\left(|0\rangle-|1\rangle\right)\left(|10\rangle+|01\rangle\right)\Big] \\ &= \frac{1}{2}\Big[|00\rangle\left(\alpha|0\rangle+\beta|1\rangle\right)+|01\rangle\left(\alpha|1\rangle+\beta|0\rangle\right) \\ &\quad +|10\rangle\left(\alpha|0\rangle-\beta|1\rangle\right)+|11\rangle\left(\alpha|1\rangle-\beta|0\rangle\right)\Big]. \end{aligned} \tag{1}$$

Dynamic quantum circuits refer to quantum circuits with mid-circuit measurements and feedforward operations [11]. Recent experimental studies have demonstrated the potential of DQCs in various application scenarios, including short-depth state preparation [27], device scale expansion [14], and quantum fan-out gate implementation [12].

The power of DQC can be illustrated by *quantum teleportation*, a technique for moving quantum states [19]. Fig. 1 shows the circuit diagram for teleporting a qubit from Alice's system (the top two lines) to Bob's system (the bottom line). Initially, Alice and Bob together generated an EPR pair and each of them takes one qubit, that is, $|\beta_{00}\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle+|11\rangle\right)$. Then the mission of Alice is to teleport an unknown state $|\psi\rangle = \alpha|0\rangle+\beta|1\rangle$ to Bob by sending only classical information, which can be achieved by the gates and measurement feedforward operations. Specifically, Alice applies a CNOT gate and a Hadamard gate to her qubits and send the measurement results of $|\psi_2\rangle$ – as derived and shown on the right side of Fig. 1 – to Bob. Depending on Alice's measurement outcome, Bob could recover the state $|\psi\rangle$ by applying corresponding operations. For example, if the measurement result is 01 then Bob can fix up his state by applying the X gate.

C. Quantum Control Systems

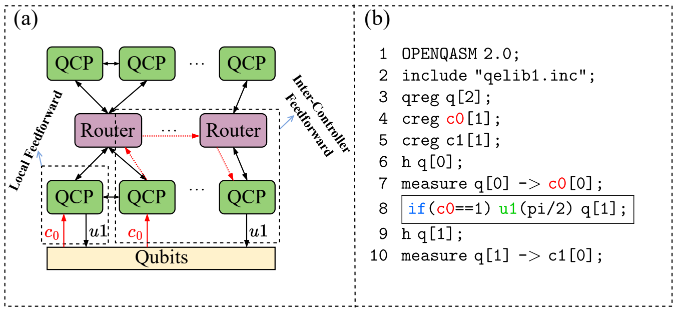
Although the immense value of DQCs has long been recognized, only recent advancements in control systems have enabled DQCs to be flexibly programmed and executed on real machines, given their stringent requirements for real-time classical processing capabilities [13], [28]. A *quantum control system* is a specialized classical system composed of hardware and software designed to control quantum devices. Quantum circuits are compiled into quantum control instructions and then executed by the control system [29]. Therefore, it is easy to comprehend the following insight:

Insight

Quantum program execution time is determined by the instruction processing time in the quantum control system.

With the number of qubits increases, quantum control systems evolve to a distributed architecture as shown in Fig. 2(a), which includes a group of QCPs interconnected via some routers [30], [31].

Taking the program shown in Fig. 2(b) as an example, performing a measurement feedforward task in the control system can involve diverse communication paths determined by the mapping between qubits and controllers. For example, the local feedforward block represents a scenario where q_0 and q_1 are managed by the same QCP. In this case, the feedforward process involves no inter-controller communication. By contrast, if qubits q_0 and q_1 are managed by separate QCPs (the inter-controller feedforward block), the measurement result C_0 may involve multiple communication hops.



```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c0[1];
creg c1[1];
h q[0];
measure q[0] -> c0[0];
if(c0==1) u1(pi/2) q[1];
h q[1];
measure q[1] -> c1[0];
```

Fig. 2: (a) Schematic of a distributed quantum control system. (b) Source program of dynamic quantum Fourier transform [12], [32].

D. Problem and Motivation

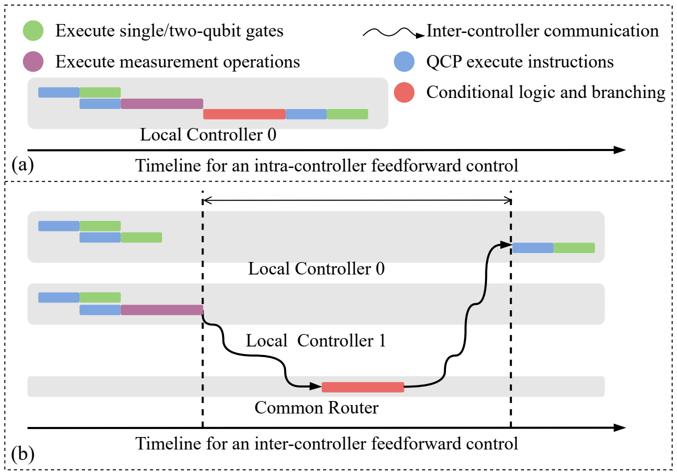

Execute single/two-qubit gates
Execute measurement operations
Inter-controller communication
QCP execute instructions
Conditional logic and branching
Local Controller 0
(a)
Timeline for an intra-controller feedforward control
Local Controller 0
Local Controller 1
Common Router
Timeline for an inter-controller feedforward control
(b)

Fig. 3: The latency breakdown for intra- and inter-controller feedforward.

Current quantum devices are constrained by limited qubit lifetimes, typically on the order of hundreds of microseconds for state-of-the-art superconducting qubits. Reducing the execution time of quantum programs to mitigate errors from decoherence has thus become a key objective. The lifecycle of a quantum control instruction can be divided into two stages based on current QCP designs [16], [29]: (i) fetching, decoding, and queuing the quantum event until it is emitted to the quantum-classical interface (QCI), represented by blue blocks; and (ii) executing pulse waveforms on quantum devices, where quantum gates are shown as green blocks and measurements, including acquisition and waiting for results, are represented by purple blocks. For static circuits without mid-circuit measurements, the execution time of quantum programs can be effectively approximated by the quantum device time, as the processing time on QCPs is pipelined with the pulse durations on the quantum device. Traditional LSQC techniques thus focus on minimizing circuit depth and SWAP overhead to reduce quantum device time. However, this approach is insufficient for DQCs, where a quantum gate instruction may depend on prior measurement results and associated classical processing and QCP time and thus cannot be well-hidden by pulse durations on the quantum device (Fig. 3(a)). Therefore, it is necessary to reshape the existing LSQC design methodology from device-centric to controller-centric.

In this work, we identify a unique optimization opportunity for the layout synthesis of DQCs. Specifically, intra-controller feedforward, where measurement and dependent qubits are managed locally by the same controller, avoids inter-controller communication, reducing latency (Fig. 3(a)). In contrast, inter-controller feedforward potentially requires communications between different controllers with extended latency (Fig. 3(b)). Existing studies validate this latency difference. For example, Ref. [33] reports a branching latency of ~500 ns in distributed systems with a central router, while intra-controller feedforward latency is as low as 92 ns [29] and can reach 50 ns in leading industry products [34]. Motivated by this discrepancy, this work aims to design a layout synthesizer that minimizes inter-controller communication latency.

III. Approach

In this section, we present the design of CLASS. We begin with *Type-I DQCs*, which are characterized by the absence of connectivity constraints and the replacement of CNOT gates with measurement and feedforward operations, significantly reducing circuit depth. A notable example is the dynamic circuit for quantum Fourier transform (QFT) [35], a core subroutine in numerous quantum algorithms [36], [37]. Next, we address *Type-II DQCs*, which are subject to connectivity constraints. For these circuits, the LSQC objectives are to minimize both inter-controller communication latency and circuit depth.

A. Type-I DQC

1) Motivational Example: To better understand the modeling approach of CLASS, we first describe an illustrational example based on a DQC-implementation of QFT as shown

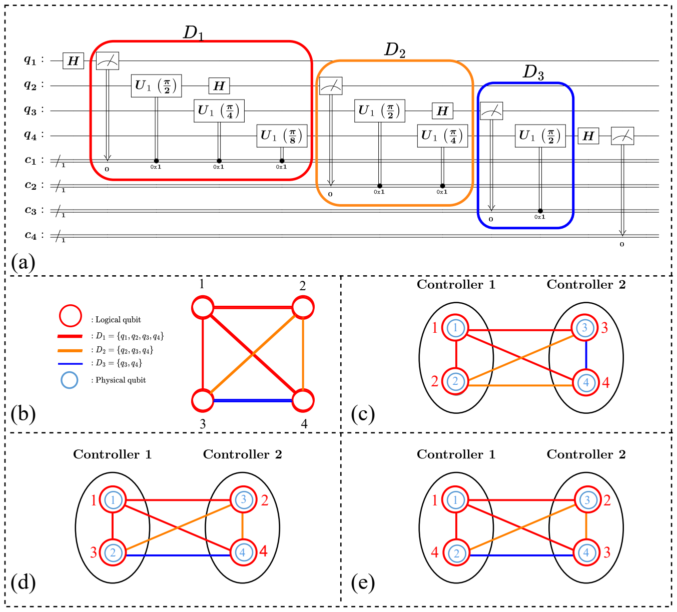


Fig. 4: An illustrational example of the minimum-cut graph-partitioning problem based on the 4-qubit DQC of QFT. (a) Circuit diagram of a 4-qubit dynamic QFT; (b) Graph representation of all feedforward operations; (c)-(e) Three different logical-to-physical qubit mapping schemes: $\mathcal{M}_1^Q, \mathcal{M}_2^Q, \mathcal{M}_3^Q$.

in Fig. 4(a). We denote logical qubits as $q_1, q_2, \ldots, q_n$, where n is the total number of qubits in the circuit.

- **Graphical representation of feedforward operations.** In Fig. 4(b), the feedforward operations from the example circuit are extracted and represented as a graph. Nodes correspond to logical qubits, and an edge between two qubits indicates that an operation on one depends on the measurement of the other. For example, the gates $\mathrm{U}_1\left(\frac{\pi}{2}\right), \mathrm{U}_1\left(\frac{\pi}{4}\right)$, and $\mathrm{U}_1\left(\frac{\pi}{8}\right)$, acting on q_2, q_3, and q_4, are all conditioned on the measurement of q_1, resulting in three red edges: $\{q_1, q_2\}, \{q_1, q_3\}, \{q_1, q_4\}$. The involved qubits form a *conditional inter-dependent qubits* (CIDQ) set, denoted as $D_1 = \{q_1, q_2, q_3, q_4\}$. Two other CIDQ sets can be similarly identified: $D_2 = \{q_2, q_3, q_4\}$ and $D_3 = \{q_3, q_4\}$. In a CIDQ set D_i, the measured qubits form D_i^m, and the target qubits that depend on those measurements form D_i^t. For instance, in D_1, we have $D_1^m = \{q_1\}$ and $D_1^t = \{q_2, q_3, q_4\}$.
- **Logical-to-physical qubit mapping schemes.** Fig. 4(c)-(e) depict different logical-to-physical qubit mapping schemes. Consider two controllers C_1 and C_2 that manage physical qubits $\{Q_1, Q_2\}$ and $\{Q_3, Q_4\}$ respectively. There is no distinction between (i) $\{q_1, q_2\} \rightarrow C_1, \{q_3, q_4\} \rightarrow C_2$ and (ii) $\{q_1, q_2\} \rightarrow C_2, \{q_3, q_4\} \rightarrow C_1$. Consequently, a little thought shows that there are only three possible logical-to-physical mappings:

$$\begin{aligned}
&\text{(i)} \quad \mathcal{M}_1^Q \equiv \{q_1, q_2\} \rightarrow \{Q_1, Q_2\}, \{q_3, q_4\} \rightarrow \{Q_3, Q_4\};\\
&\text{(ii)} \quad \mathcal{M}_2^Q \equiv \{q_1, q_3\} \rightarrow \{Q_1, Q_2\}, \{q_2, q_4\} \rightarrow \{Q_3, Q_4\};\\
&\text{(iii)} \quad \mathcal{M}_3^Q \equiv \{q_1, q_4\} \rightarrow \{Q_1, Q_2\}, \{q_2, q_3\} \rightarrow \{Q_3, Q_4\}.
\end{aligned} \tag{2}$$

Our goal is to find a logical-to-physical mapping that minimizes inter-controller communication. Under $\mathcal{M}_1^Q$, the first (D_1) and second (D_2) feedforward operations require sending measurement results of q_1 and q_2 from C_1 to C_2, while the third (D_3) is performed locally on C_2. Thus, $\mathcal{M}_1^Q$ incurs 2 communication steps between C_1 and C_2. In comparison, $\mathcal{M}_2^Q$ and $\mathcal{M}_3^Q$ each result in 3 communication steps. Therefore, $\mathcal{M}_1^Q$ is the optimal mapping with the lowest communication overhead. Interestingly, as shown in Fig. 4(c)-(e), the number of distinct edge colors crossing the two controllers matches the number of communication steps. This observation forms the basis of our approach: transforming the LSQC problem into an equivalent graph-cut problem:

Core Idea

Layout synthesis for minimizing inter-controller communication latency is equivalent to a minimum-cut graph partitioning problem.

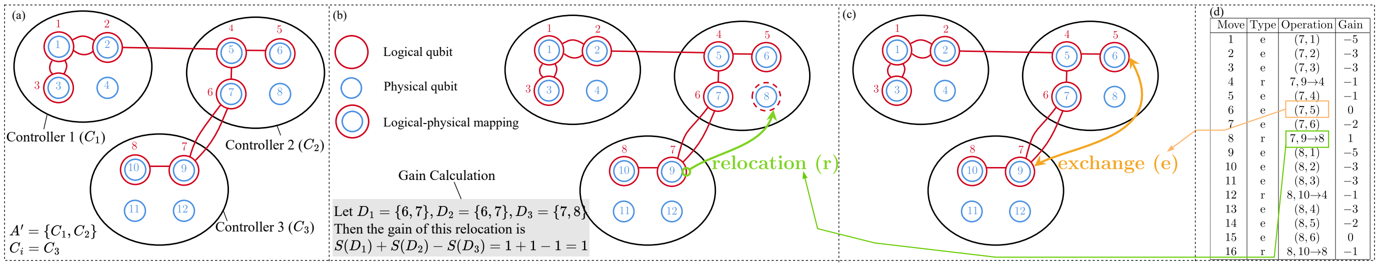
2) Problem Formulation: The minimum-cut graph partitioning problem can be formally described as follows.

- **Hypergraph definition.** Given a DQC, each feedforward operation is extracted as a CIDQ set, forming a list L^D, where $L^D[i, j]$ denotes the jth qubit in the ith CIDQ set D_i. Using L^D, we construct a hypergraph $U(V, E)$ representing all feedforward dependencies, as illustrated in Fig. 4(b). Each node in V corresponds to a logical qubit, and each hyperedge in E represents a CIDQ set connecting multiple nodes.
- **Graph partitioning.** Consider a distributed quantum control system with k controllers $\{C_i\}_{i=1}^k$ managing m physical qubits $\{Q_i\}_{i=1}^m$. Each controller is responsible for a subset of physical qubits, defined by a mapping function $\mathcal{M}^C \equiv f : \{Q_i\} \rightarrow \{C_j\}$. The hypergraph U is partitioned into subgraphs, each assigned to a distinct controller. As a result, some hyperedges are cut across multiple controllers. Note that, a specific partitioning scheme is determined by the logical-to-physical qubit mapping $\mathcal{M}^Q$, which is the output of our layout synthesizer.
- **Optimization objective.** Cut hyperedges correspond to inter-controller communication. Our objective is finding a mapping $\mathcal{M}^Q$ to minimize the total communication latency, defined as the sum of latencies associated with all cut edges.

$$\mathcal{L}(\mathcal{M}^Q) \equiv \sum_{D_i \in L^D} S(D_i). \tag{3}$$

Here, $S(D_i)$ denotes the communication latency for sending measurement results from D_i^m to D_i^t, which depends on the topology and communication protocol of the control system. In our implementation, we use the minimum number of communication hops—determined by the controller topology—to represent this latency, which we refer to as *Inter-Controller Communication Steps* (ICCS).

3) Algorithm Design: While the example in Fig. 4 partitions graph vertices into equally sized subsets, our problem



Move	Type	Operation	Gain
1	e	$(7, 1)$	-5
2	e	$(7, 2)$	-3
3	e	$(7, 3)$	-3
4	r	$7, 9 \to 4$	-1
5	e	$(7, 4)$	-1
6	e	$(7, 5)$	0
7	e	$(7, 6)$	-2
8	r	$7, 9 \to 8$	1
9	e	$(8, 1)$	-5
10	e	$(8, 2)$	-3
11	e	$(8, 3)$	-3
12	r	$8, 10 \to 4$	-1
13	e	$(8, 4)$	-3
14	e	$(8, 5)$	-2
15	e	$(8, 6)$	0
16	r	$8, 10 \to 8$	-1

Fig. 5: Examples of different movements and gain calculation during a qubit moving pass (line 7 in Alg. 2). For illustration purpose, we assume all CIDQ sets involve only two qubits and the ICCS between any pair of controllers is uniformly 1. (a) A possible logical-to-physical mapping $\mathcal{M}^Q_{current}$. (b) Example of qubit relocation and its gain calculation process. After relocating q_7 to Q_8, D_1 and D_2 belong to the same controller, while D_3 is cut by two controllers. Therefore, the gain is calculated by $1 + 1 - 1 = 1$. (c) Example of qubit exchange. (d) Table of gains of all movements between C_3 and $A' = \{C_1, C_2\}$.

TABLE I: List of symbols used in our approach.

Symbol	Description
D_i	A specific CIDQ set representing a group of conditionally interdependent qubits.
D_i^m	Subset of qubits in D_i that are measured.
D_i^t	Subset of qubits in D_i that require measurements from other qubits.
$\mathcal{M}^Q$	Logical-to-Physical qubit mapping function: Maps logical qubits to physical qubits on a device.
$\mathcal{M}^C$	Qubit-Controller mapping function: Maps physical qubits to their controlling QCPs.
L_D	A list of CIDQ sets derived from a given DQC.
$S(D_i)$	Communication cost for executing feedforward operations in D_i.
$\mathcal{L}(\mathcal{M}^Q)$	Objective function to minimize: The sum of $S(D_i)$ over all CIDQ sets.
U	Undirected hypergraph where vertices represent logical qubits and hyperedges represent CIDQ sets, used to transform the LSQC problem into a graph partitioning task.

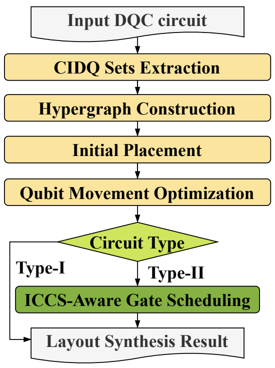


Fig. 6: The layout synthesis flow of CLASS.

does not require such balance. For instance, in a DQC with 12 qubits and two controllers, each managing up to 10 qubits, assigning the two qubits involved in the fewest CIDQ sets to one controller and the remaining 10 qubits to the other may yield better results than an even split. This differentiates our problem from traditional k-way graph-partitioning problems [38]. To address this, we propose a two-stage heuristic approach (Alg. 1), detailed as follows.

- **Stage 1: Initialize $\mathcal{M}^Q$ via greedy allocation of controllers.** First, we construct an undirected graph U based on L^D, where the vertices in U correspond to logical qubits in a DQC. The graph U is a *hypergraph*, allowing edges to connect multiple vertices, with each CIDQ set D_i represented as a hyperedge in U. Next, we traverse the vertices in U in descending order of their *degrees* – defined as the number of CIDQ sets in which a qubit is included – to *prioritize the allocation of highly interdependent qubits to*

Algorithm 1: ICCS-Aware Initial Placement

```
Input  : Controller-Qubit Mapping $\mathcal{M}^C$, List of CIDQ Sets $L^D$
Output: Logical-Physical Mapping $\mathcal{M}^Q$
// Stage 1: Initialize $\mathcal{M}^Q$
1  $U$ ←construct_graph($L^D$);
2  Initialize $\mathcal{M}^Q(q_i)$ as -1 for all qubits;
3  foreach logical qubit $q_i$ in the descending order of degrees in $U$ do
4      $controller\_score \leftarrow \{\}$;
5      foreach $q_j$ in the neighbors of $q_i$ in $U$ do
6          $Q_j \leftarrow \mathcal{M}^Q(q_j)$;
7          if $Q_j \neq -1$ then
8              $C_j \leftarrow \mathcal{M}^C(Q_j)$;
9              $controller\_score[C_j] \leftarrow controller\_score[C_j] + 1$;
10     if controller_score is not empty then
11         Find $C_i$ with the maximum score in $controller\_score$;
12     else
13         Randomly choose a controller $C_i$ for $q_i$ and its neighbors;
14     Randomly choose a physical qubit $Q_i$ from the physical qubits obtained
         from the inverse mapping of $\mathcal{M}^C$;
15     $\mathcal{M}^Q$.update($q_i \rightarrow Q_i$);
// Stage 2: Iteratively move qubits across the
   allocated controllers to further reduce the number
   of ICCSs
16 $best\_score \leftarrow \infty$;
17 foreach controller $C_i$ in all $k$ controllers $A \equiv \{C_a\}_{a=1}^k$ do
18     $A' \leftarrow A \backslash \{C_i\}$; // Obtain the controllers excluding
         $C_i$
19     $\mathcal{M}^Q_{temp} \leftarrow$ apply_qubit_moving_pass($\mathcal{M}^Q, C_i, A'$);
20     $current\_score \leftarrow \mathcal{L}(\mathcal{M}^Q_{temp})$;
21     if $current\_score < best\_score$ then
22         $\mathcal{M}^Q \leftarrow \mathcal{M}^Q_{temp}$;
23         $best\_score \leftarrow current\_score$ ;
```

Algorithm 2: Qubit Moving Pass

```
Input  : Logical-Physical Mapping $\mathcal{M}^Q$, Controller $C_i$, Set of Other
         Controllers $A'$
Output: New Logical-Physical Mapping $\mathcal{M}^Q_{temp}$
1  $\mathcal{M}^Q_{temp} \leftarrow \mathcal{M}^Q$;
2  $\mathcal{M}^Q_{current} \leftarrow \mathcal{M}^Q$;
3  $best\_movements \leftarrow []$;
4  $gains \leftarrow []$;
5  $movements$ ←obtain_movements($C_i, A'$);
6  while $movements \neq \emptyset$ do
7      Find a $move$ that has the maximal gain $g$ based on $\mathcal{M}^Q_{current}$;
8      $movements$.remove($move$);
9      $best\_movements$.append($move$);
10     $gains$.append($g$);
11     $\mathcal{M}^Q_{current}$.update($move$);
12 Find a $l$ that maximize $max\_gain \leftarrow \sum_{j=1}^{l} gains[i]$;
13 if $max\_gain > 0$ then
14     $\mathcal{M}^Q_{temp}$.update($best\_movements[1:l]$);
```

the same controller. For each vertex (qubit), if its neighbors have not been assigned to any controller, we randomly map this qubit and its neighbors to the physical qubits of a controller. Otherwise, we identify the controllers managing its neighboring qubits and map it to the controller responsible for the most neighbors (lines 2-14).

- **Stage 2: Iteratively move qubits across the allocated controllers to further reduce the number of ICCSs.** Let $A \equiv \{C_a\}_{a=1}^{k}$ denote the set of all k controllers. For each controller C_i in A, we exclude C_i to form a new set A' (line 18) and perform a *qubit moving pass* between C_i and A', generating a temporary mapping $\mathcal{M}_{temp}^{Q}$ (line 19). The mapping with the lowest objective function value $\mathcal{L}(\mathcal{M}_{temp}^{Q})$ is selected as the final mapping (lines 20–23). Inspired by the classic Kernighan–Lin (KL) algorithm [39], which searches between two partitions, our approach (Alg. 2) generalizes this by exploring movements between one controller (C_i) and all remaining controllers (A'), thereby significantly expanding the search space. Furthermore, unlike KL's bidirectional exchanges, our *hybrid movement design* includes both bidirectional exchanges of qubits between C_i and A' (Fig. 5(c)), as well as unidirectional relocations of a single qubit from C_i to another controller in A', provided this does not exceed the capacity constraints of the target controller (Fig. 5(b)). This design is enabled by the absence of a balance constraint in our problem, thus the relocation of a single qubit allows our algorithm to explore solution spaces of unbalanced partitions. The *gain* of a movement, defined as the change in the sum of ICCSs of the affected CIDQ sets, is shown in Fig. 5(d), based on the initial mapping in Fig. 5(a). In this example, a unidirectional relocation achieves the highest gain, demonstrating the effectiveness of incorporating hybrid movements and highlighting the potential benefits of removing balance constraints. In each iteration, we iteratively select, apply, and remove the movement with the greatest gain until all are considered (lines 6–11). We then update $\mathcal{M}_{temp}^{Q}$ with the maximum accumulated gain and finalize the mapping (lines 12–14).

4) Complexity Analysis: In Stage 1, for each qubit (with n in total), we traverse its neighboring qubits (with d on average) and identify the controllers allocated to these qubits. Then, we determine the controller that manages the largest number of neighboring qubits among all k controllers. Thus, the complexity of Stage 1 is $O(n(k+d))$. In Stage 2, for each controller (with k in total), we perform a qubit moving pass, where the complexity is determined by the number of movements. The average number of qubits managed by a controller is $\frac{n}{k}$. Since each qubit can either be moved from C_i to any of the other controllers in A' or exchanged with any other qubit in A', the number of movements per qubit is $(n+k)$. Therefore, the total number of movements per pass is $N = \frac{n}{k}(n+k)$. These movements can be stored in a priority queue, with the movement of maximum gain ($move$) placed at the front. After applying $move$, the gains of movements associated with the qubits involved in $move$ need to be recalculated. The total number of such associated movements is $M = d(n+k)$. Since updating an element in a priority queue has a complexity of $O(\log(N))$, the complexity of updating gains is $O(M \cdot \log(N))$. Finally, the total complexity of stage 2 is $O(k \cdot N \cdot M \cdot \log(N)) = O(k \cdot \frac{n}{k}(n+k) \cdot d(n+k) \cdot \log(\frac{n}{k}(n+k))) = O(dn(n+k)^2 log(\frac{n}{k}(n+k)))$.

B. Type-II DQCs

Connectivity constraints add complexity to minimizing inter-controller communication. Since two-qubit gates require physically adjacent qubits, SWAP gates are often inserted to enable gate execution—a process known as *gate scheduling* [3] or *qubit routing* [21]. Prior work has largely focused on reducing circuit depth or the number of SWAPs, without considering the underlying controller architecture. However, inserting SWAPs may inadvertently split a CIDQ set across multiple controllers, increasing # ICCS.

To address this challenge, we design a latency-aware gate scheduler that minimizes both the number of additional SWAP gates and inter-controller latency. The core idea is to *use inter-controller latency as a tie-breaker when multiple SWAP insertion options have the same cost.* Existing heuristic layout synthesizers, such as those in Refs. [2], [6], [21], typically use greedy search strategies that evaluate potential SWAP costs at each step by considering future SWAP insertion possibilities based on predefined cost functions. In some cases, multiple SWAP options may yield identical or nearly identical costs. This creates an opportunity to incorporate a controller-latency metric to proactively guide SWAP selection, avoiding random

Algorithm 3: ICCS-Aware Gate Scheduling

```
Input  : Controller-Qubit Mapping $\mathcal{M}^C$, List of CIDQ Sets $L^D$, Initial
         Logical-Physical Mapping $\mathcal{M}^Q$, Front Layer $F$, Device Coupling
         Map $G(V, E)$
Output: Inserted SWAPs, Final Logical-Physical Mapping $\mathcal{M}_f^Q$
1  while $F$ is not empty do
2      $execute\_gate\_list \leftarrow \emptyset$;
3      Find executable gates in $F$ and put them into $execute\_gate\_list$;
4      if $execute\_gate\_list \neq \emptyset$ then
5          foreach gate in $execute\_gate\_list$ do
6              Remove gate from $F$ and put its successors in DAG into $F$ if
                 the gates' dependencies are resolved;
7          continue;
8      else
9          $score = \{\}$;
10         $swap\_candidate\_list$ = obtain_swaps($F, G$);
11         for $swap$ in $swap\_candidate\_list$ do
12             $\mathcal{M}_{temp}^Q = \mathcal{M}^Q$.update($swap$);
13             $score[swap] \leftarrow$
                 obtain_depth_cost($F, DAG, G, \mathcal{M}_{temp}^Q$);
14         $similar\_swaps \leftarrow$obtain_min_score_swaps($score$);
           // ICCS-Aware Search
15         if len($similar\_swaps$) $> 1$ then
16             $iccs\_score \leftarrow \{\}$;
17             for $swap$ in $similar\_swaps$ do
                   // Calculate ICCS score for these SWAPs
18                 $\mathcal{M}_{temp}^Q \leftarrow \mathcal{M}^Q$.update($swap$);
19                 $L_{active}^D \leftarrow$
                     obtain_cidq_sets($F, DAG, G, \mathcal{M}_{temp}^Q$);
20                 $iccs\_score[swap] \leftarrow \sum_{D_i \in L_{active}^D} S(D_i)$;
21             $swaps \leftarrow$ obtain_min_score_swaps($iccs\_score$);
22             $swap \leftarrow$ random_choice($swaps$);
23         else
24             $swap \leftarrow similar\_swaps[0]$;
25         $\mathcal{M}^Q$.update($swap$);
```

choices. Notably, this design philosophy can be generalized to all existing gate schedulers, enabling them to account for controller communication latency during the decision-making process of whether and where to insert SWAPs. As a demonstration, we integrate our strategy into the gate scheduling stage of SABRE [2] as it is widely used in the community. Alg. 3 describes the process of ICCS-aware SABRE search, where the directed acyclic graph (DAG) represents the execution dependencies of operations in the circuit. The front layer F is defined as the set of all two-qubit gates with no unexecuted predecessors in the DAG, and the coupling map G represents the topology of the target quantum device. For a comprehensive explanation of its modeling methodology, readers may refer to the original SABRE paper [2]. Here, we omit some details of the original SABRE steps and focus on explaining the relevant modifications that make it aware of ICCS. When there are no executable gates in F, we first identify all possible SWAPs associated with the qubits in F and calculate a score for each SWAP to estimate its negative impact on circuit depth (lines 9∼13). Next, we initiate an ICCS-aware search procedure when multiple SWAPs yield identical scores (lines 15–22). For each SWAP, we first obtain a temporary mapping $\mathcal{M}^{Q}_{temp}$ by applying the SWAP. We then look ahead to identify a set of feedforward operations whose dependencies in the DAG are resolved after applying the SWAP and extract the corresponding CIDQ sets from L^D to construct a new list, L^D_{active}. Subsequently, we calculate the sum of the ICCSs associated with each CIDQ set in L^D_{active} to determine the $iccs\_score$ of the SWAP. The SWAP with the lowest $iccs\_score$ is selected as the final choice.

IV. Evaluation

A. Experiment Setup

1) Implementation: We implement CLASS as a framework that interfaces with Qiskit. CLASS consists of ∼3k lines of Python code and around ∼1k lines of modifications in the Rust library of Qiskit. For Type-I DQCs, we set the outcome of our initial placement as the initial layout of Qiskit transpiler. For Type-II DQCS, we extend the SABRE implementation in Qiskit and use the initial placement as the starting layout.

2) Benchmarks: Benchmarks are collected from VeryQBench [32] and QASMBench [40], including dynamic QFT, iterative phase estimation (PE), and the quantum counterfeit coin (CC) problem. Additionally, we construct randomized DQCs to cover a broader range of circuit patterns by using the blocks from the randomized benchmarking protocol [15] as basic components (Random).

3) Metrics: Several post-compilation metrics are collected for performance comparison with Qiskit as our baseline, including circuit depth, number of operations, and number of ICCSs (# ICCS).

4) System Configurations: In our main results, we adopt a star-topology controller architecture, where all controllers are connected to a central router [30]. This topology is chosen as, to the best of our knowledge, it is the only publicly available solution that provides both a concrete controller topology design and support for arbitrary feedforward operations. The qubit device architecture is based on IBM's 127-qubit quantum processor, which features a heavy-hex-lattice topology. All experiments were performed on a Linux server with 768 GB of memory and two 32-core Intel(R) Xeon(R) Silver 4216 CPUs.

TABLE II: Comparison between CLASS and baseline ($k = 4$).

Benchmark	**Qubits**	Baseline			CLASS		
		# Operations	Depth	# ICCS	# Operations	Depth	# ICCS
Type-I Benchmarks							
qft	20	270	99	144	270	99	0
qft	30	555	149	335	555	149	0
qft	40	940	199	599	940	199	256
qft	50	1425	249	933	1425	249	576
Type-II Benchmarks							
cc	12	159	109	24	153	110	6
cc	32	487	315	120	586	346	6
pe	20	441	171	125	434	184	19
pe	30	798	268	309	870	277	29
pe	40	1356	400	591	1444	430	309
pe	50	1938	505	911	2101	505	619
random	20	2096	726	339	2095	785	39
random	30	5551	1589	882	5738	1852	111
random	40	11480	2810	1500	11413	3016	912
random	50	19743	4710	2691	20140	4621	1812
Type-II Average	-	4404.90	1160.30	749.20	4497.40	1212.60	386.20

B. Performance on Type-I DQCs

To the best of our knowledge, no prior work has addressed the same problem as the one we tackle in this study. All existing layout synthesizers fall back to generate a randomized layout for Type-I DQCs. In contrast, CLASS is explicitly designed to account for controller-architectural constraints, aiming to minimize # ICCS. As shown in Table II, CLASS achieves a significant reduction in # ICCS while keeping the number of operations and circuit depth unchanged. Notably, for QFT circuits with 20 and 30 qubits, CLASS achieves 100% reduction of # ICCS, as the number of qubits is small enough to be mapped entirely within a region managed by a single controller. As shown in Fig. 7, for QFT circuits with 40 and 50 qubits, CLASS reduces # ICCS by 57.26% and 38.26%, respectively.

C. Performance on Type-II DQCs

For DQCs with connectivity constraints, we also achieve considerable reductions in inter-controller communication latency, while introducing only a small overhead in terms of the number operations and circuit depth. As summarized in Table II, the post-compilation circuits of the baseline approach and CLASS contain an average of 4404.9 and 4497.4 operations, respectively. This indicates that CLASS introduces only a modest overhead of approximately 2.10% in additional CNOT gates. In contrast, CLASS reduces the average number of ICCSs from 749.20 to 386.20, representing a substantial reduction of approximately 48.45%, as shown in Fig. 8.

D. Impact of Controller Architecture

We have also verified the adaptability of CLASS to arbitrary controller architecture. Specifically, we generate various controller topologies with randomized and diverse inter-controller communication latency. Subsequently, we transpile a 12-qubit QFT circuit onto these architectures and compare the performance between CLASS and baseline. On average, we obtain 46.71% reduction in terms of # ICCS.

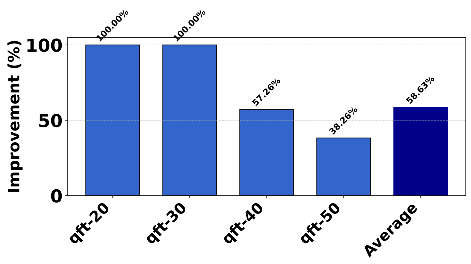


Fig. 7: ICCS reduction for Type-I DQCs.

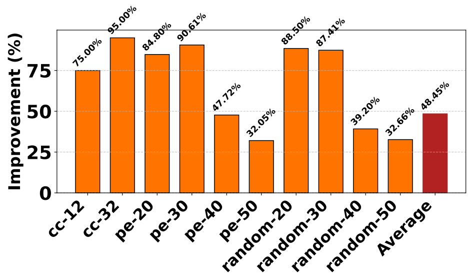


Fig. 8: ICCS reduction for Type-II DQCs.

Additionally, we conduct a study to evaluate the impact of the number of controllers (denoted as k) on the performance of CLASS. For a randomized DQC with 30 qubits, we vary k from 4 to 8. As shown in Fig. 9, the improvement of CLASS over the baseline decreases as k increases. This occurs because increasing k leads to a higher number of subgraphs in the graph-partitioning problem, naturally resulting in more edge cuts between these subgraphs. In an extreme scenario where each controller manages only a single qubit, inter-controller communications cannot be eliminated, and our approach would show no improvement over the baseline.

E. Scalability Analysis

Since our gate scheduler extends existing schedulers, its runtime is primarily influenced by those schedulers. Thus we focus our evaluation on the performance of our initial placement algorithm. As analyzed in Sec. III-A, the complexity of our algorithm exhibits polynomial growth with respect to the number of qubits and controllers. In practical quantum computing systems, the number of controllers is typically fixed. Therefore, we set $k = 5$ and vary the number of qubits in dynamic QFT circuits to profile the runtime of our initial placement algorithm. As shown in Fig. 10, the runtime increases from approximately 4 seconds to 7 seconds as the number of qubits grows from 20 to 100. The efficiency of our algorithm could be further enhanced by employing more efficient system-level programming languages.

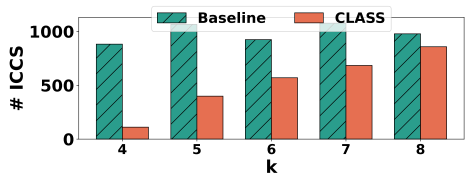


Fig. 9: Impact of the number of controllers.

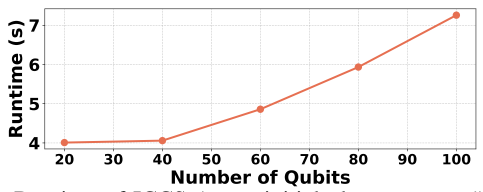


Fig. 10: Runtime of ICCS-Aware initial placement vs. # qubits.

V. Discussion

A. Feasibility of CLASS in Quantum Error Correction

The feasibility of applying CLASS in future QEC scenarios depends on whether topological constraints persist in next-generation devices. Since many QEC protocols, such as surface codes [41], are inherently designed to align with device topology, SWAP-based qubit routing may no longer pose a major challenge. In this context, CLASS may not be directly applicable to QEC applications. Nevertheless, we believe the controller-centric design methodology remains relevant due to the continued importance of feedforward operations.

B. Feasibility of CLASS in Real Systems

It is currently challenging to deploy and evaluate CLASS on real quantum systems. CLASS requires a distributed quantum control architecture that supports arbitrary feedforward operations, but the design of such systems remains an active research area [31], [42]. Although industrial platforms such as IBM Quantum Cloud can execute arbitrary DQCs, their control systems are not publicly accessible. As a result, evaluating the performance of CLASS on public quantum cloud platforms is currently infeasible. To overcome this limitation, developing an in-house control system becomes necessary—an effort that is currently underway.

VI. Conclusion and Future Work

This work addresses the challenges posed by inter-controller communication delays in the layout synthesis of dynamic quantum circuits (DQCs). For DQCs without connectivity constraints, we model the problem as a minimum-cut task in an undirected hypergraph and solve it using an efficient heuristic approach. This solution provides the initial placement for DQCs with connectivity constraints. To mitigate the impact of SWAP insertions during gate scheduling, we enhance existing schedulers with an effective mechanism. Our evaluations demonstrate that our synthesizer achieves up to a 100% reduction in inter-controller communications, with only a $\sim$2% increase in additional CNOTs.

Acknowledgments

We thank Dr. Xiang Fu for his valuable feedback and insightful discussions. This work is supported in part by National Natural Science Foundation of China (NSFC) (62025404, 62222411, 62304037), National Key Research and Development Program of China (2023YFB4404400), and the Natural Science Foundation of Jiangsu Province under Grant BK20230828.

References

[1] R. Wille, L. Burgholzer, S. Hillmich, T. Grurl, A. Ploier, and T. Peham, "The basis of design tools for quantum computing: arrays, decision diagrams, tensor networks, and zx-calculus," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1367–1370.

[2] 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.

[3] B. Tan and J. Cong, "Optimal layout synthesis for quantum computing," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[4] W.-H. Lin, J. Kimko, B. Tan, N. Bjørner, and J. Cong, "Scalable optimal layout synthesis for nisq quantum processors," in *Design Automation Conference (DAC)*, 2023.

[5] R. Wille and L. Burgholzer, "Mqt qmap: Efficient quantum circuit mapping," in *Proceedings of the 2023 International Symposium on Physical Design*, 2023, pp. 198–204.

[6] 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.

[7] S. Park, D. Kim, M. Kweon, J.-Y. Sim, and S. Kang, "A fast and scalable qubit-mapping method for noisy intermediate-scale quantum computers," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 13–18.

[8] R. Wille, L. Burgholzer, and A. Zulehner, "Mapping quantum circuits to ibm qx architectures using the minimal number of swap and h operations," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[9] W. Tang, Y. Duan, Y. Kharkov, R. Fakoor, E. Kessler, and Y. Shi, "Alpharouter: Quantum circuit routing with reinforcement learning and tree search," *arXiv preprint arXiv:2410.05115*, 2024.

[10] J. Yang, Y. A. Kharkov, Y. Shi, M. J. Heule, and B. Dutertre, "Quantum circuit mapping based on incremental and parallel sat solving," in *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024, pp. 29–1.

[11] A. D. Córcoles, M. Takita, K. Inoue, S. Lekuch, Z. K. Minev, J. M. Chow, and J. M. Gambetta, "Exploiting dynamic quantum circuits in a quantum algorithm with superconducting qubits," *Physical Review Letters*, vol. 127, no. 10, p. 100501, 2021.

[12] E. Bäumer, V. Tripathi, A. Seif, D. Lidar, and D. S. Wang, "Quantum fourier transform using dynamic circuits," *Physical Review Letters*, vol. 133, no. 15, p. 150602, 2024.

[13] IBM. (2022) Bringing the full power of dynamic circuits to qiskit runtime. https://www.ibm.com/quantum/blog/quantum-dynamic-circuits. [Online]. Available: https://www.ibm.com/quantum/blog/quantum-dynamic-circuits

[14] A. C. Vazquez, C. Tornow, D. Riste, S. Woerner, M. Takita, and D. J. Egger, "Scaling quantum computing with dynamic circuits," *arXiv preprint arXiv:2402.17833*, 2024.

[15] L. Shirizly, L. C. Govia, and D. C. McKay, "Randomized benchmarking protocol for dynamic circuits," *arXiv preprint arXiv:2408.07677*, 2024.

[16] X. Fu, M. A. Rol, C. C. Bultink, J. Van Someren, N. Khammassi, I. Ashraf, R. Vermeulen, J. De Sterke, W. Vlothuizen, R. Schouten, C. Almudéver, L. DiCarlo, and K. Bertels, "An experimental microarchitecture for a superconducting quantum processor," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 813–825.

[17] IBM. (2025) IBM quantum roadmap. https://www.ibm.com/roadmaps/quantum/. [Online]. Available: https://www.ibm.com/roadmaps/quantum/

[18] Google. (2025) Google quantum computing roadmap. https://quantumai.google/roadmap. [Online]. Available: https://quantumai.google/roadmap

[19] M. A. Nielsen and I. L. Chuang, "Quantum computation and quantum information," 2010. [Online]. Available: https://doi.org/10.1017/CBO9780511976667

[20] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: a case for variability-aware policies for nisq-era quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 987–999.

[21] H. Fu, M. Zhu, J. Wu, W. Xie, Z. Su, and X.-Y. Li, "Effective and efficient qubit mapper," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[22] A. Molavi, A. Xu, M. Diges, L. Pick, S. Tannu, and A. Albarghouthi, "Qubit mapping and routing via maxsat," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1078–1091.

[23] S.-T. Huang, Y.-J. Jiang, S.-Y. Fang, and C.-K. Cheng, "Smt-based layout synthesis for silicon-based quantum computing with crossbar architecture," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–8.

[24] I. Shaik and J. van de Pol, "Optimal layout synthesis for quantum circuits as classical planning," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[25] N. Quetschlich, L. Burgholzer, and R. Wille, "Compiler optimization for quantum computing using reinforcement learning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[26] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross *et al.*, "Quantum computing with qiskit," *arXiv preprint arXiv:2405.08810*, 2024.

[27] K. C. Smith, A. Khan, B. K. Clark, S. Girvin, and T.-C. Wei, "Constant-depth preparation of matrix product states with adaptive quantum circuits," *PRX Quantum*, vol. 5, no. 3, p. 030344, 2024.

[28] A. Cross, A. Javadi-Abhari, T. Alexander, N. De Beaudrap, L. S. Bishop, S. Heidel, C. A. Ryan, P. Sivarajah, J. Smolin, J. M. Gambetta, and B. R. Johnson, "Openqasm 3: A broader and deeper quantum assembly language," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, pp. 1–50, 2022.

[29] X. Fu, L. Riesebos, M. Rol, J. Van Straten, J. Van Someren, N. Khammassi, I. Ashraf, R. Vermeulen, V. Newsum, K. Loh *et al.*, "eqasm: An executable quantum instruction set architecture," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 224–237.

[30] G. Zettles, S. Willenborg, B. R. Johnson, A. Wack, and B. Allison, "26.2 design considerations for superconducting quantum systems," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 1–3.

[31] F. Zhang, X. Zhu, R. Chao, C. Huang, L. Kong, G. Chen, D. Ding, H. Feng, Y. Gao, X. Ni *et al.*, "A classical architecture for digital quantum computers," *ACM Transactions on Quantum Computing*, vol. 5, no. 1, pp. 1–24, 2023.

[32] K. Chen, W. Fang, J. Guan, X. Hong, M. Huang, J. Liu, Q. Wang, and M. Ying, "Veriqbench: A benchmark for multiple types of quantum circuits," *arXiv preprint arXiv:2206.10880*, 2022.

[33] R. S. Gupta, N. Sundaresan, T. Alexander, C. J. Wood, S. T. Merkel, M. B. Healy, M. Hillenbrand, T. Jochym-O'Connor, J. R. Wootton, T. J. Yoder *et al.*, "Encoding a magic state with beyond break-even fidelity," *Nature*, vol. 625, no. 7994, pp. 259–263, 2024.

[34] Z. Instruments. (2024) Quantum feedback measurements. https://www.zhinst.com/japan/en/applications/quantum-technologies/quantum-feedback-measurements. [Online]. Available: https://www.zhinst.com/japan/en/applications/quantum-technologies/quantum-feedback-measurements

[35] R. B. Griffiths and C.-S. Niu, "Semiclassical fourier transform for quantum computation," *Physical Review Letters*, vol. 76, no. 17, p. 3228, 1996.

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

[37] A. Paler, "Quantum fourier addition simplified to toffoli addition," *Physical Review A*, vol. 106, no. 4, p. 042444, 2022.

[38] W. L. Lee, D.-L. Lin, T.-W. Huang, S. Jiang, T.-Y. Ho, Y. Lin, and B. Yu, "G-kway: multilevel gpu-accelerated k-way graph partitioner," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[39] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.

[40] 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*, vol. 4, no. 2, pp. 1–26, 2023.

[41] 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.

[42] G. Huang, Y. Xu, N. Fruitwala, A. D. Rajagopala, K. Nowrouzi, R. K. Naik, D. Santiago, and I. Siddiqi, "Qubic 2.0: A flexible advanced full stack quantum bit control system," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 2. IEEE, 2023, pp. 248–249.