









AdaptDQC: Adaptive Distributed Quantum Computing With Quantitative Performance Analysis

Debin Xiang , Liqiang Lu , Siwei Tan , Xinghui Jia , Zhe Zhou , Guangyu Sun , *Senior Member, IEEE*,
Mingshuai Chen , and Jianwei Yin , *Senior Member, IEEE*

Abstract—We present AdaptDQC, an adaptive compiler framework for optimizing distributed quantum computing (DQC) under diverse performance metrics and inter-chip communication (ICC) architectures. AdaptDQC leverages a novel spatial-temporal graph model to describe quantum circuits, model ICC architectures, and quantify critical performance metrics in DQC systems, yielding a systematic and adaptive approach to constructing circuit-partitioning and chip-mapping strategies that admit hybrid ICC architectures and are optimized against various objectives. Experimental results on a collection of benchmarks show that AdaptDQC outperforms state-of-the-art compiler frameworks: It reduces, on average, the communication cost by up to 35.4% and the latency by up to 38.4%.

Index Terms—Distributed quantum computing, heterogeneous quantum communication, distributed systems.

I. INTRODUCTION

QUANTUM computing has undergone a recent surge of interest due to its potential of efficiently solving problems that classical computers struggle with. Prominent instances include prime factorization [1], database search [2], approximate optimization [3], simulation of complex systems [4], machine learning [5], and cryptography [6]. Nonetheless, efficiently solving large-scale quantum computing tasks is beyond the capability of the current noisy intermediate-scale quantum (NISQ) era [7] due to certain technical defects [8]: The presence of complex noise, e.g., crosstalk [9] and analog noise [10], limits the number of qubits on a single chip up to 500; The extremely short decoherence time [11] further restricts the scale of quantum gates up to 1000 layers.

The paradigm of distributed quantum computing (DQC) [12], [13], [14] is deemed a promising solution to

TABLE I
COMPARISON AMONG EXISTING DQC SOLUTIONS

Configurations		CutQC [16]	Hyper-Graph [19]	Auto-Comm [20]	AdaptDQC
ICC arc.	QubitComm	✓	✗	✗	✓
	GateComm	✗	✓	✓	✓
Opt. obj.	Comm.	✓	✓	✓	✓
	Latency	✗	✗	✓	✓
	Topology	✗	✗	✗	✓

significantly improve the scalability of quantum computing, where a monolithic quantum circuit – encoding a quantum computation task – is partitioned into subcircuits and thereafter distributed across multiple inter-connected quantum chips.

One crucial aspect of implementing DQC is inter-chip communication (ICC), which can be achieved through two main approaches: *qubit communication* (QubitComm) and *gate communication* (GateComm). QubitComm amounts to partitioning quantum circuits along qubits and then deploying them onto different quantum chips; Chips then exchange qubit information via probabilistic reconstruction utilizing tensor computation [15], [16], [17]. In contrast, GateComm divides quantum circuits per gate and deploys the so-obtained subcircuits onto different chips; Remote gates acting on different chips communicate via quantum teleportation [18], which relies on quantum measurement and classical links.

As various DQC infrastructures adopting the aforementioned ICC architectures have been recently proposed, e.g., CutQC [16], HyperGraph [19], and AutoComm [20] optimizing the overall performance of these systems becomes a prominent challenge: We argue that none of the existing DQC frameworks per se can fulfill the high-performance DQC requirements (cf. Table I) due to three reasons.

First, existing frameworks focus on a single type of ICC architecture and lack a unified optimization strategy applicable to different systems: CutQC employs the MIP algorithm model [21] to achieve circuit partitioning along qubit wires as QubitComm; HyperGraph leverages the hyper-graph model [22] to distribute gates to different chips per GateComm; Based on HyperGraph, AutoComm further incorporates block transportation of qubit-chip gates to reduce the communication overhead. However, considering the distinct advantages of different ICC architectures, a systematic, unified compiler framework

Received 7 June 2024; revised 23 January 2025; accepted 1 July 2025. Date of publication 14 July 2025; date of current version 10 September 2025. This work was supported in by National Key Research and Development Program of China under Grant 2023YFF0905200 and in part by Zhejiang Provincial Natural Science Foundation of China under Grant LR25F020002 and Grant LD24F020013. Recommended for acceptance by J. Palsberg. (Corresponding authors: Liqiang Lu; Jianwei Yin.)

Debin Xiang, Liqiang Lu, Siwei Tan, Xinghui Jia, Mingshuai Chen, and Jianwei Yin are with the College of Computer Science, Zhejiang University, Hangzhou 310058, China (e-mail: db.xiang@zju.edu.cn; liqianglu@zju.edu.cn; siweitian@zju.edu.cn; jxhjh@zju.edu.cn; m.chen@zju.edu.cn; zjuyjw@cs.zju.edu.cn).

Zhe Zhou and Guangyu Sun are with the College of Computer Science, Peking University, Beijing 100871, China (e-mail: zhou.zhe@pku.edu.cn; gsun@pku.edu.cn; lj@pku.edu.cn).

Digital Object Identifier 10.1109/TC.2025.3586027

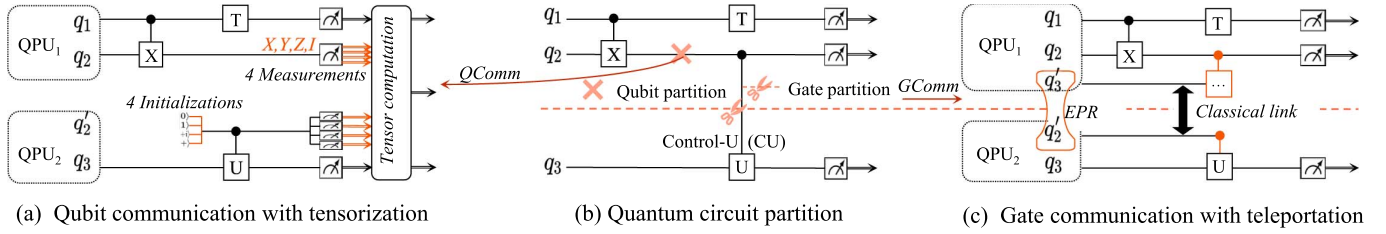


Fig. 1. ICC architectures: (a) QubitComm uses tensor computation to achieve communication between QPUs. (b) Partition quantum circuit by qubit partition or gate partition. (c) QubitComm uses teleportation to implement the remote control-unitary gate CU for communication between QPUs.

admitting different architectures can potentially lead to significant performance improvement.

Second, there is a lack of a unified, analytical model to evaluate the DQC system performance. CutQC evaluates the number of partitions; HyperGraph assesses the communication cost between chips, and AutoComm considers system communication and latency. Although these performance metrics have different emphases, they cannot comprehensively signify the performance of DQC frameworks so as to guide the design.

Third, existing frameworks are primarily limited to optimizing one (or two) individual performance metrics. For instance, CutQC and HyperGraph aim to minimize the communication cost whereas AutoComm focuses on minimizing the teleportation latency. In practice, however, a DQC framework is usually expected to be able to (1) support multiple performance metrics simultaneously and (2) adjust its optimization strategy adaptively in terms of different objectives.

In response to these limitations, we propose three requirements that a comprehensive DQC framework should meet:

- R1: unified description.** The framework should incorporate a unified approach to describing DQC systems with diverse ICC architectures and performance metrics.
- R2: quantitative performance evaluation.** To guide the optimization in compiling different DQC systems, the framework should provide a quantitative performance evaluation mechanism to cope with diverse metrics.
- R3: adaptive optimization strategy.** The framework should adjust its optimization strategy adaptively based on specific conditions, thus ensuring optimal performance across different quantum tasks and ICC architectures.

This paper contributes a novel compiler framework, AdaptDQC, that meets *all* the three requirements above (cf. Table I). Specifically, to fulfill **R1**, we propose a spatial-temporal graph model to describe DQC systems employing different ICC architectures. Such a model consists of a spatial directed hypergraph (SDHG), a temporal directed acyclic graph (TDAG), and a chip-level graph (CG), which accurately capture the spatial information flow (carried by qubits), the temporal information flow (processed by gates), and the inter-chip communication (implemented by ICC) respectively in quantum computing. We further exploit the spatial-temporal graph model to quantify the performance of distributed quantum systems, thereby addressing **R2**. We consider the following performance metrics: spatial metrics (i.e., topology cost), temporal metrics (i.e., latency), and inter-chip metrics (i.e., communication cost), which are

quantified via the SDHG, TDAG, and CG, respectively. These quantified metrics can be subsequently used to evaluate the quantitative performance of different DQC systems and guide their optimization. Finally, to meet **R3**, we design an adaptive compiler that features graph clustering algorithms to optimize different ICC architectures. AdaptDQC is capable of automatically selecting the optimal compilation strategy based on user requirements, thereby improving the performance for various optimization goals. To summarize, our main contributions are as follows:

- We model DQC systems via novel spatial-temporal graphs, facilitating the application of several graph-optimization techniques to DQC compilation (Section IV-A).
- Our spatial-temporal graph model supports quantitative performance analysis of DQC systems in terms of spatial, temporal, and inter-chip metrics (Section IV-B).
- We present the unified, adaptive DQC compiler AdaptDQC to optimize multiple performance metrics under diverse ICC architectures (Section V). To the best of our knowledge, AdaptDQC is the first DQC framework that admits hybrid ICC architectures.

We demonstrate the superiority of AdaptDQC in position to several baselines via a comprehensive collection of quantum-algorithm benchmarks (Section VI): Our compiler, when optimized towards latency, can reduce the average latency by 42.5% in the QubitComm architecture compared to CutQC [16], and 18.2% in the GateComm architecture compared to HyperGraph [19] and AutoComm [20]. When optimized towards communication, AdaptDQC reduces 15.4× and 26.3% communication costs in QubitComm and GateComm, respectively. The experiments further show that the hybrid ICC architecture reduces 35.4% communication overhead and 38.4% latency compared to single architectures.

II. BACKGROUND AND MOTIVATION

A. Distributed Quantum Computing Basis

Distributed quantum computing (DQC) has been proposed to expand the scale of quantum computing. It divides large quantum circuits into smaller subcircuits, which can be implemented on different quantum processing units (QPUs). For example, in Fig. 1(b), the original quantum circuit can be partitioned into two subcircuits (cf. Fig. 1(a)) along the horizontal line of qubit q_2 . Similarly, partitioning the vertical line of CU gate can also split the circuit (cf. Fig. 1(c)). These subcircuits are then

distributed to small quantum chips [12], [15]. Compared to a large, monolithic quantum chip, small chips have exponentially reduced noise levels, leading to higher fidelity [23]. The existing implementations of DQC differ primarily in the adopted inter-chip communication (ICC) architectures. These ICC architectures can be classified into two categories: *Qubit Communication* and *Gate Communication*, which are introduced as follows:

Qubit-Communication (QubitComm). In this ICC architecture, after deploying the subcircuits onto different quantum chips, the communication between chips is realized via probabilistic reconstruction utilizing tensor computation [15]. As illustrated in Fig. 1(a), after partitioning the original circuit along q_2 , we apply measurements on four bases I, X, Y, Z on the qubit before partition point and four initialization $|0\rangle, |1\rangle, |+\rangle, |+i\rangle$ on the qubit after partition point, respectively. The four measurements and four initializations contain all the information of the partitioned qubit. After getting the probability distribution of each subcircuit executed on quantum chips, we then rebuild the whole distribution through Equation 1.

$$p(|m\rangle) = \sum_{\Phi \in \{|0\rangle, |1\rangle, |+\rangle, |+i\rangle\}} \frac{1}{2} (p_{\Phi} \otimes p(|m\rangle|\Phi)) \quad (1)$$

Here, p_{Φ} denotes the probability of qubit state Φ in the subcircuit being measured. $p(|m\rangle|\Phi)$ denotes the probability of $|m\rangle$ in the subcircuit initialized by Φ . Then the probabilistic reconstruction is achieved by classical tensor computation. In this paper, we name it *qubit communication with tensorization*. For more theoretical details, please refer to [15].

Gate-Communication (GateComm). Another approach for communication between quantum chips is through the implementation of remote entanglement gates. As shown in Fig. 1(b), if we partition the original circuit along the CU gate, then in Fig. 1(c), QPU_1 executes q_1, q_2 , while QPU_2 executes q_3 . The control qubit q_2 of CU is assigned to QPU_1 while the target qubit q_3 is in another QPU, making CU become a remote gate between two QPUs. The remote gates between quantum chips are realized by quantum teleportation or cat-entanglement [24], which teleports or links the control qubit to another quantum chip. In this paper, we name it *gate communication with teleportation*. In the quantum teleportation process, one pair of entangled qubits, namely, an EPR pair [25], are prepared and distributed on separated QPUs, such as q'_3 on QPU_1 and q'_2 on QPU_2 . In this manner, we can apply entangled gates between q_2 and q'_3 on QPU_1 with a classical link to teleport q_2 to q'_2 . Since q'_2 and the target qubit q_3 are in the same chip, we can apply CU between them to achieve gate communication.

B. Motivation

Prior works primarily focus on single architectures: CutQC [16] supports only QubitComm, while HyperGraph [19] and AutoComm [20] are limited to GateComm. This separation arises because QubitComm relies on tensor networks, whereas GateComm depends on teleportation. However, these single architectures may consume more quantum hardware resources.

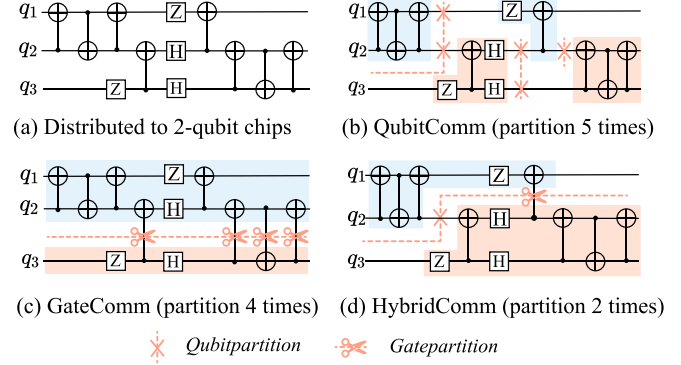


Fig. 2. Motivation: a hybrid ICC will reduce system communication cost compared with the single ICC.

For example, Fig. 2 shows an example where we aim to distribute a three-qubit quantum circuit (cf. Fig. 2(a)) on the two-qubit quantum hardware device. Fig. 2(b) gives the minimal partitions on the QubitComm architecture. Here, we need to partition the original circuit at least five times to obtain four two-qubit subcircuits. Fig. 2(c) shows the GateComm architecture case. Four CNOT gates are partitioned, making the original circuit into two pieces. In summary, in this case, applying either the GateComm or QubitComm method needs to partition the original circuit at least four times, leading to more communication overhead.

To address this, we propose a hybrid architecture (HybridComm) that combines QubitComm and GateComm. By first partitioning circuits using QubitComm and then distributing subcircuits via GateComm, HybridComm reduces the number of partitions and communication overhead. For example, the same three-qubit circuit can be distributed with only two partitions in HybridComm (Fig. 2(d)).

Moreover, HybridComm offers a practical solution for the NISQ era. QubitComm incurs high post-processing overhead due to tensor network computations, while GateComm suffers from low teleportation fidelity (92% [26]), which requires resource-intensive entanglement distillation [27] to improve. HybridComm balances these trade-offs, minimizing post-processing overhead while maintaining computational accuracy, making it a flexible and practical approach for distributed quantum computing.

III. FRAMEWORK OVERVIEW

As various ICC architectures have been proposed, designing an adaptive high-performance distributed quantum computing framework becomes more challenging. We have to address requirements **R1-R3** introduced in Section I. To meet these requirements, we propose an adaptive DQC compiler framework, named AdaptDQC.

As illustrated in Fig. 3, AdaptDQC takes a large quantum circuit (①) as input and outputs compiled distributed circuits (⑥) according to the underlying DQC system information (⑤). AdaptDQC contains three core parts:

- The DQC formulation gives the formal description of the DQC system. It comprises three graph-level models:

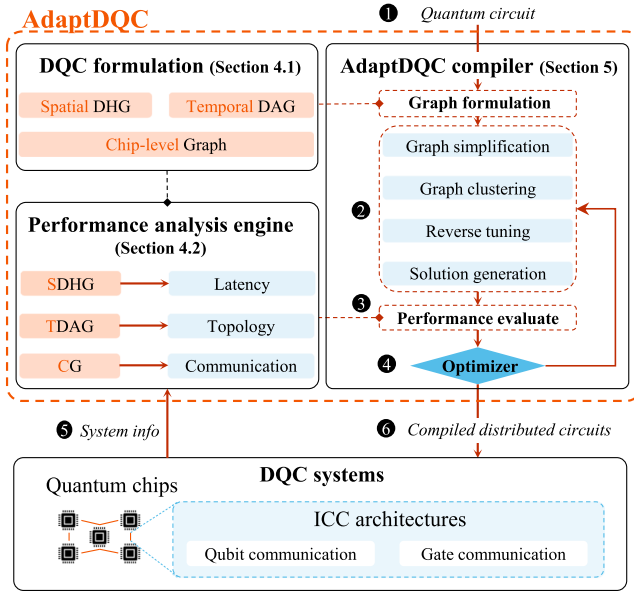


Fig. 3. AdaptDQC framework.

Spatial Directed Hyper-Graph (SDHG), the *Temporal Directed Acyclic Graph* (TDAG), and the *Chip-level Graph* (CG). The SDHG regards each qubit as a node, which serves to model the spatial relation in DQC systems and facilitates the evaluation of spatial metrics. The TDAG formulates each gate as a node, which helps to model the temporal relation. The CG regards each partitioned subcircuit chip as a node, which depicts the overall DQC systems at the chip-level.

- Based on the Graph formulation, our performance analysis engine enables quantitative pre-evaluation of the compiled circuits. It uses SDHG to quantify spatial performance metrics (i.e., topology), uses TDAG to quantify temporal metrics (i.e., latency), and uses CG to quantify chip-level metrics (i.e., communication).
- The AdaptDQC compiler takes quantum circuits as input and then uses various optimization strategies to compile the distributed quantum circuits. The compilation process consists of circuit partition, performance evaluation, and optimization. During circuit partition (2), we first convert the circuit into TDAG and propose an adaptive clustering algorithm for QubitComm, GateComm, and HybridComm. The adaptive clustering algorithm consists of four steps: graph simplification, graph clustering, reverse tuning, and solution generation. According to the clustered DQC solutions, the performance analysis engine then evaluates the system performance (3), which provides optimization guidance for the optimizer (4). Finally, the compiler derives the optimized subcircuits (6), which will be deployed on DQC systems.

The following sections provide more details. We first introduce the graph models and quantitative performance analysis in Section IV. Finally, Section V describes how the graph and performance models are used to build the Adaptive DQC compiler.

IV. DQC FORMULATION

In quantum computing, qubits and quantum gates play distinct roles in information processing. Qubits encode and carry information in space, while quantum gates manipulate this information over time. DQC involves distributing qubits and quantum gates across different chips to decompose computational tasks in the temporal and spatial domains. However, this decomposition introduces communication overheads in both domains. To facilitate the representation of spatial and temporal communication, we propose a graph-based approach decoupled in temporal and spatial dimensions. In this context, graphs consist of nodes representing information units (e.g. gates or qubits) and edges denoting the relationships between the nodes. We use the *Spatial Directed Hyper-Graph* (SDHG) with qubits as nodes to formulate the spatial relationships in DQC and the *Temporal Directed Acyclic Graph* (TDAG) with gates as nodes to capture the temporal relationships in DQC. The overall DQC solution is formulated as chip-level graph (CG). Fig. 4(a) is an example of hybrid distributed quantum computing, which partitions a 3-qubit circuit and deploys the pieces into three QPUs.

A. Modeling in Spatial and Temporal Dimension

Definition IV.1: SDHG is defined by converting qubits into nodes and gates into edges. The number of nodes gives the spatial capacity of DQC systems, i.e., the number of qubits in each partitioned subcircuit. In particular, SDHG duplicates the partitioned qubit for the probabilistic reconstruction in QubitComm and introduces two qubits for the EPR in GateComm. The edges are the gates in the original circuit, such as CX, CU.

We use the SDHG to model the spatial relations in DQC systems. The SDHG captures the inter-qubit relations, such as two-qubit gate operations. Fig. 4(b) shows an example of SDHG derived from Fig. 4(a). As the qubit line of q_2 is partitioned, the SDHG duplicates this qubit and distributes it to two QPUs for QubitComm. And the edge between these two nodes means the tensorization operation. On the other hand, the gate $CU[q_2, q_3]$ is partitioned resulting in GateComm. Thus, SDHG generates two separated nodes as EPR between two QPUs, where the edge represents the teleportation communication.

Definition IV.2: TDAG is defined by converting gates into nodes and qubits into edges. The edge connects the gate node to another gate node in order, denoting the execution sequence of gates, which implies the temporal relationship in DQC systems. In particular, TDAG introduces two additional teleportation nodes to bridge the entanglement between QPUs for GateComm. For QubitComm, TDAG formulates the tensorization as an edge between two corresponding QPUs.

Except for the spatial relations, we need to model the temporal relationships to capture the execution dependency. Thus, we introduce TDAG to achieve this goal. Fig. 4(c) gives an example of TDAG that describes the execution dependency of gates. According to the partition in Fig. 4(a), TDAG models the tensorization operation as an edge between CX node in QPU3 and H gate in QPU2. Besides, the partition on the

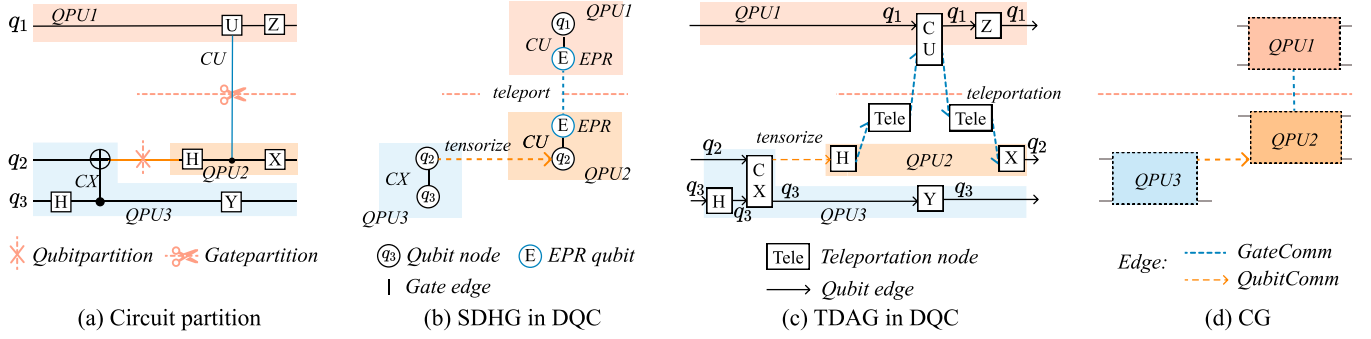


Fig. 4. The graph formulation of DQC: (a) circuit partition by qubit and gate. (b) SDHG converted from the partitioned circuit. (c) TDAG converted from the partitioned circuit. (d) CG of the partitioned circuit.

TABLE II
METRICS OF DQC IMPLEMENTATION

Metrics	GateComm	QubitComm
Topology	$d(G_{subcircuits}, G_{chips})$	
Latency	$\sum_{n \in \text{the longest path}} \text{Latency}(n)$	$\max_{g \in G} \text{Latency}(g)$
Communication	$\sum_{c \in G} \#Q(c) + \sum_{e \in E(G)} \#Q_{EPR}$	$\sum_{c \in G} 4^{\ E(c)\ } \#Q(c)$

TABLE III
LATENCY (IN NS) OF DQC OPERATIONS

Operation	Symbol	Super Conductor	Ion Traps	Neutral Atoms
single-qubit gate	t_{1q}	10	5,000	2,667
two-qubit gate	t_{2q}	22	120,000	11,370
measure	$t_{measure}$	10	100,000	80,000
teleport	t_{tele}	100	220,000	130,000

gate $CU[q_2, q_3]$ leads to two teleportation nodes that are used to connect $QPU1$ and $QPU2$.

Definition IV.3: CG is defined as a hyper-graph that models each quantum chip as a node, where the edge connects the chip nodes, representing the communication mechanism among them, i.e., QubitComm and GateComm.

We use CG to depict the overall DQC systems at the chip-level, which can be extracted from either SDHG or TDAG by merging the nodes in the same QPU, as shown in Fig. 4(d). The tensorization from $QPU3$ to $QPU2$ is preserved as the QubitComm edge, and the teleportation between $QPU2$ to $QPU1$ is preserved as the GateComm edge.

B. Performance Analysis Engine

The performance analysis engine is extended from the defined graph models. It quantitatively estimates critical metrics including the compatibility between distributed circuits topology and physical chip layout (topological mapping overhead), the time delay in processing (latency), and the amount of data transferred between chips (communication cost). These metrics are very important because they affect the reliability of the results when deploying on distributed quantum chips. Generally, lower values for these metrics mean more reliable outcomes. Topological mapping overhead is naturally a spatial feature that can be derived from the SDHG model. While latency involves temporal features, e.g., the latency of a circuit shows forward-increment along the sequence of gates. Thus, it can be modeled via the TDAG. The communication cost is a chip-level metric that can be evaluated from CG. We have gathered all these metrics and displayed them in Table II for easy reference.

Topological mapping overhead refers to transforming the subcircuits to fit the topology of the target quantum chips.

The quantitative latency of operations is extracted from previous research [29].

Since the SDHG can also describe the topology of the quantum chip, we can model this overhead T by calculating the distance between two graphs:

$$T = d(G_{subcircuit}, G_{chip})$$

$$d(G_1, G_2) = \min_{\phi} \sum_{(v,u) \in G_1} (S_{G_2}(\phi(u), \phi(v)) - 1) \quad (2)$$

where, $G_{subcircuit}$ is the SDHG of the partitioned subcircuit, and G_{chip} is the topology graph of the target chip represented in SDHG. d is the distance between two graphs. Here, $\phi: V(G_1) \rightarrow V(G_2)$ is the mapping for the nodes from G_1 to G_2 . $S_{G_2}(\phi(u), \phi(v))$ is the length of the shortest path [28] between node $\phi(u)$ and $\phi(v)$. A zero distance means that two graphs are perfectly matched. Nonzero distance indicates an unmatched topology, resulting in the SWAP gate cost of qubit mapping. Thus, higher topological mapping overhead will degrade the computational fidelity and increase the overall latency.

Latency can be evaluated by the TDAG with node latency. Table III lists the operations and their latency in the DQC system. The latency of a subcircuit is calculated by summing the latency of the gates in the longest path. For the QubitComm architecture, in the quantum execution step, we assume the subcircuits can be executed in parallel. This is because, in most circumstances, the number of hardware resources is enough to parallel the distributed subcircuits. Thus, the overall latency in quantum execution is the maximum latency among subcircuits.

$$L_{QubitComm} = \max_g \text{Latency}(g),$$

$$g \in \text{the subcircuits of TDAG} \quad (3)$$

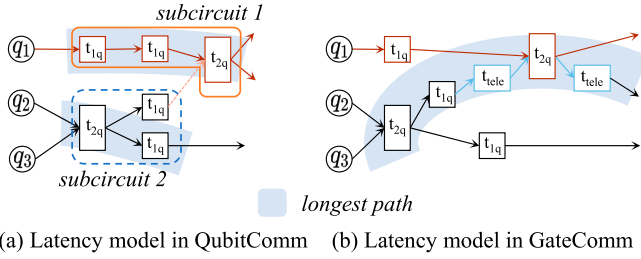


Fig. 5. Formulating latency using the TDAG: (a) latency in QubitComm is the max latency among subcircuits. (b) Latency in GateComm is the longest path.

Here, $Latency(g)$ is the latency of the subcircuit g in TDAG. For example, in Fig. 5(a), the latency of the DQC system is determined by the longest latency among subcircuits, i.e., $2t_{1q} + t_{2q}$.

For the GateComm case, subcircuits are executed sequentially according to the dependency in the TDAG. The total latency of GateComm is calculated by summing the latency in the longest path of TDAG.

$$L_{GateComm} = \sum_{n \in \text{the longest path of TDAG}} Latency(n), \quad (4)$$

For example, in Fig. 5(b), the latency of the GateComm case is $2t_{2q} + t_{1q} + 2t_{tele}$.

Communication cost is defined as the amount of communication data between chips in CG. The number of qubits serves as the fundamental unit for measuring quantum information. Both QubitComm and GateComm require additional qubit resources to facilitate inter-chip communication. Specifically, QubitComm consumes qubit resources by running circuits multiple times and GateComm needs more qubits to prepare EPR pairs. Thereby, we use the number of consumed qubits to quantify the communication cost.

CG captures the communication pattern between chips via edges, so the communication cost can be calculated from these edges. We first group the chips of CG by QubitComm edges, denoted by subgraph g . Different subgraphs g communicate with each other only by QubitComm. Inside subgraph g , chip nodes communicate by GateComm. The total communication cost is to sum the number of qubits spent on each subgraph g

$$Comm = \sum_{g \in G_{QubitComm}} \#Qubit(g) \quad (5)$$

To calculate the consumed qubits resource of subgraph $Qubits(g)$, we first consider QubitComm. We need to run the circuits of subgraph g for $O(4^{K(g)})$ times when cutting $K(g)$ qubits on a subgraph g . Thus, the consumed qubits resource of each subgraph g is multiplied by a factor $4^{K(g)}$ to model the copies of circuits due to QubitComm.

$$\#Qubit(g) = 4^{K(g)} \cdot \#Q_{GateComm}(g) \quad (6)$$

Here, the number of circuit cuts $K(g)$ is calculated from the number of all edges connected to the subgraph g ,

$$K(g) = \|\{g \in e | e \in E(G_{QubitComm})\}\|$$

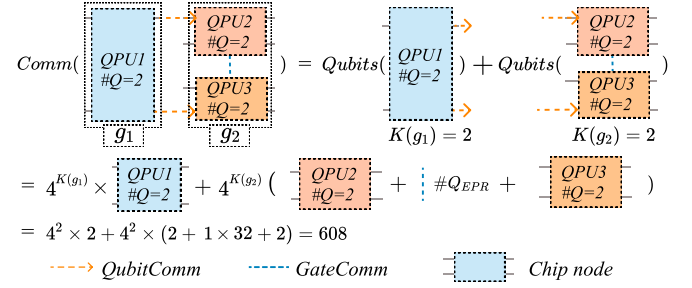


Fig. 6. Formulating communication using the CG.

The resource cost $\#Q_{GateComm}(g)$ inside the subgraph g is composed of the qubits resource on each chip node and the extra qubits to build GateComm between chips. The qubits resource on each chip node c is computed by the maximum width of the chip $\#Q(c)$. For GateComm, we denote the number of extra qubits for preparing an EPR pair as $\#Q_{EPR}$, which is 32 in a current superconducting quantum chip by entanglement purification [27]. Thus, the total consumed qubits with GateComm $\#Q_{GateComm}(g)$ in a subgraph g is

$$\#Q_{GateComm}(g) = \sum_{c \in g} \#Q(c) + \sum_{e \in E(g)} \#Q_{EPR} \quad (7)$$

Fig. 6 gives an example to calculate the communication cost using CG. The communication cost of this example is $4^2 \#Q_{QPU1} + 4^2 (\#Q_{QPU2} + 1 \cdot \#Q_{EPR} + \#Q_{QPU3})$.

V. ADAPTDQC COMPILER

Our Adaptive Compiler is specifically designed to facilitate adaptive distributed computing, leveraging performance metrics from the analysis engine. Because DQC is formulated as a mapping from TDAG to CG, the compiler clusters the TDAG into a CG to generate the DQC solution with optimization goals tailored for the specific application. These optimization goals encompass crucial factors such as reducing topology mapping overhead, minimizing latency, and optimizing communication costs. The resulting CG represents subcircuits that can be executed on distributed quantum chips. The overall workflow of the Adaptive Compiler (rf. Fig. 7) consists of four steps: (a) graph simplification, (b) graph clustering, (c) adaptive reverse tuning, and (d) solution generation.

Graph simplification simplifies the TDAG by merging the continuous single-qubit or two-qubit gates. It aims to reduce the complexity of the graph by analyzing the gate dependency. Most quantum gates exhibit a single dependency, such as single-qubit gates, which can be merged into a single node. Thus, we define a set of gates \mathcal{S} that only have one forward ancestor gate:

$$\mathcal{S} = \{v \mid \forall v \in TDAG, \|\{u \mid (u, v) \in E(TDAG)\}\| = 1\} \quad (8)$$

Here, $E(TDAG)$ denotes the set of all edges in $TDAG$, and $\{u \mid (u, v) \in E(TDAG)\}$ represents all the forward ancestors of gate v . Each gate in \mathcal{S} is merged into its forward ancestor in the TDAG.

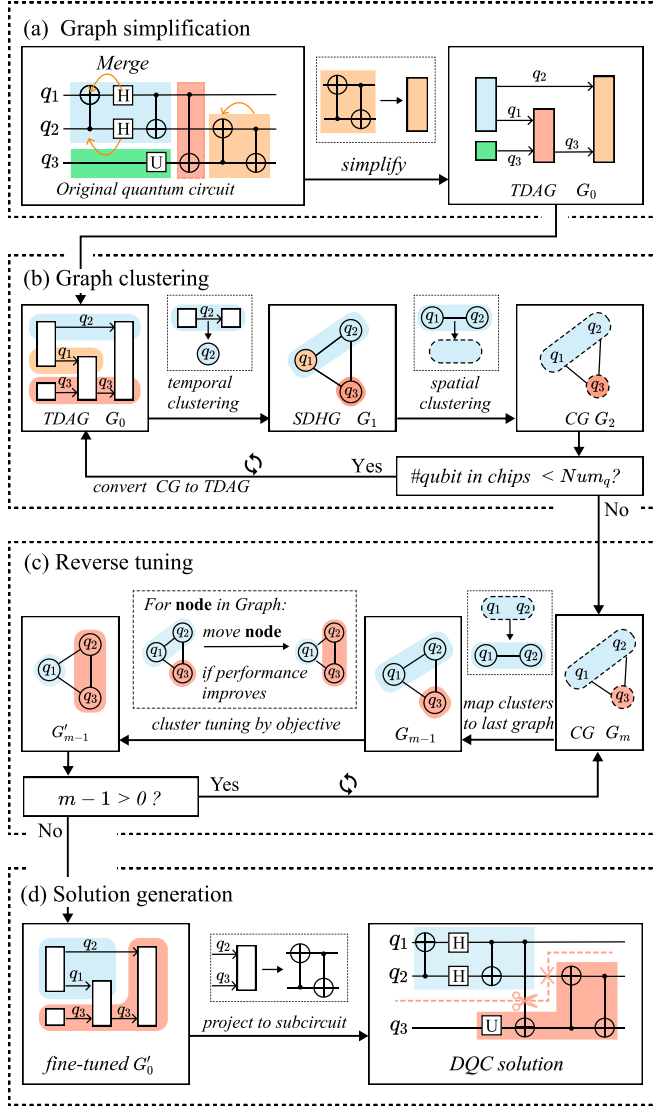


Fig. 7. Adaptive compiler algorithm: (a) simplify quantum circuit into TDAG. (b) Cluster graph to form an initial DQC solution. (c) Tune the clusters in the graphs reversely to get a better solution. (d) Generate the optimal solution from the well-tuned graph.

As illustrated in Fig. 7(a), the merged gates set S is labeled by four boxes with different colors. For example, the single-qubit gates such as $H[q_1]$ and $H[q_2]$ are merged, as they have only one forward ancestor $CX[q_2, q_1]$ (blue box). Similarly, the two-qubit gate $CX[q_2, q_3]$ is merged into its only forward ancestor $CX[q_3, q_2]$ (orange box). Note that the $CNOT[q_1, q_3]$ has two forward ancestors, and $U[q_3]$ has no ancestors, thus they cannot be simplified. Merging the gates set S gives the TDAG G_0 . This simplification reduces the depth of the TDAG and preserves the structure of the original quantum circuit.

Graph clustering iteratively clusters the TDAG in the temporal and spatial domains, which is used to generate an initial DQC solution based on the simplified TDAG G_0 . We use a temporal-spatial clustering method to form an initial solution. Temporal clustering is a conversion from TDAG to SDHG, which clusters the edges into a new qubit node and converts the

nodes into edges. Spatial clustering is a conversion from SDHG to CG, which clusters nodes with different qubits in SDHG into a new chip node in CG. The temporal-spatial clustering clusters the nearest neighbors, following a greedy strategy. We iteratively apply the temporal-spatial clustering to increase the qubits of chips in CG, until the maximum number of available qubits in quantum chips Num_q is reached. The graphs generated in every clustering step, denoted by G_0, G_1, \dots, G_M , are saved for further fine-tuning.

For example, in Fig. 7(b), the temporal scheme clusters edges with qubits q_1, q_2, q_3 in TDAG G_0 into new nodes, labeled as $\textcircled{q_1}$, $\textcircled{q_2}$, and $\textcircled{q_3}$ in SDHG G_1 . The spatial scheme clusters node $\textcircled{q_1}$ and node $\textcircled{q_2}$ in SDHG G_1 into a new node in CG G_2 . Then we repeat the temporal and spatial clustering until the number of qubits in the chips of CG reaches Num_q . We send the generated graphs in each clustering step, labeled as $G_0, G_1, G_2, \dots, G_M$, to the next step for tuning the partition of the graph.

Reverse tuning aims to tune the initial DQC solution for a better solution according to the performance metrics. The final CG G_M does not necessarily represent the optimal partition. Therefore, we tune the clusters based on predefined performance metrics (Section IV-B). The tuning on the clusters is done in reverse, i.e., from G_m to G_{m-1} . In each iteration of reverse tuning, we first map the clusters in graph G_m to the latest graph G_{m-1} . This mapping will generate the initial clustering results in G_{m-1} . Then we tune the clusters in G_{m-1} by the heuristic inspired by the Fiduccia-Mattheyses algorithm [30]. For each chip node n in graph G_m , we calculate the performance change Δ when moving a node n to another cluster c by $\Delta(n, c) = f(\text{move } n \text{ to } c) - f(\text{no movement})$. We then select the cluster c that maximizes the $\Delta(n, c)$ if $\Delta(n, c) > 0$, i.e., $c = \text{argmax}_c \Delta(n, c)$ and move n to cluster c . This process is repeated for every node in the graph until the $\Delta(n, c) \leq 0$, which means any movement will cause a performance degradation. This process will optimize the initial graph G_m into a graph G'_m with optimal performance. The mapping and tuning process will repeat until reaching the most fine-grained graph G_0 .

For example, in Fig. 7(c), we map the clusters of G_m to the last graph G_{m-1} , thus G_{m-1} is partitioned into two clusters, $\{\textcircled{q_1}, \textcircled{q_2}\}$ and $\{\textcircled{q_3}\}$. Then we start tuning the cluster in G_{m-1} by the performance metrics. For instance, we try to move $\textcircled{q_2}$ to another cluster $\{\textcircled{q_3}\}$. If the required performance metric improves, we accept the movement, and G_{m-1} is updated to two clusters with $\{\textcircled{q_1}\}$ and $\{\textcircled{q_2}, \textcircled{q_3}\}$. Tuning graph G_{m-1} results in graph G'_{m-1} . We repeat this process for the next graph G_{m-2} until we get the well-tuned CG G_0 .

Solution generation generates the final DQC solution from the fine-tuned CG G'_0 . Each cluster in G'_0 represents a subcircuit. The edges between clusters indicate communications between chips. We reverse the mapping from *Graph simplification* to reconstruct the subcircuits and identify the partition positions, thus completing the adaptive compilation process. For instance, in Fig. 7(d), the node with q_2, q_3 can be reconstructed as two CNOT gates. With this method, we get the DQC solution as shown in the last box of Fig. 7(d).

TABLE IV
BENCHMARKS FOR EVALUATION

Abbrev.	Description	#qubit	#two-qubit gate
AE	Amplitude Estimation [31]	30-126	90-4194
DJ	Deutsch-Jozsa [32]	7-126	27-125
GHZ	GHZ State [33]	13-129	12-128
GRPH	Graph State [34]	13-129	13-129
QAOA	Quantum Approximation Optimization Algorithm [35]	10-100	10-40
QNN	Quantum Neural Network [36]	10-100	32-392
ROUT	Vehicle Routing Problem [37]	6-15	25-78
VQE	Variational Quantum Eigensolver [38]	10-50	1780-4998
WSTT	Preparing W States [39]	30-80	28-98

VI. EXPERIMENTS

A. Experimental Settings

Benchmarks. We evaluate the performance of AdaptDQC on various quantum algorithms listed in Table IV. The #qubit column denotes the spatial volume of a quantum circuit. The #two-qubit gate column indicates the temporal volume of computation. The maximum width of the quantum process unit(QPU) is set to $\{5, 10, 16, 27, 50, 65\}$, according to the configurations of real-world quantum processors [40].

Baselines. We compare AdaptDQC against baseline frameworks under QubitComm and GateComm architectures. To be specific, for QubitComm, we compare AdaptDQC with CutQC [16], the state-of-the-art circuit-cutting compiler adopting the MIP model [21]. Considering that current research mainly focuses either on qubit allocation or teleportation optimization, we combine the hypergraph [19] partition for qubits allocation and teleportation optimization from Autocomm [20] to form the baseline, which is hereafter labeled as “Hyper+Autocomm”. In this way, for GateComm, we are able to compare AdaptDQC with “Hyper+Autocomm”. To establish the baselines for comparison, we adopt the open-sourced code of CutQC [16] under the QubitComm architecture. For GateComm architectures, since the Hypergraph [19] and Autocomm [20] methods lack open-source code, we faithfully re-implement the algorithms described in their papers.

Implementation. AdaptDQC is developed using IBM’s Qiskit toolkit [41]. We also utilize the rustworkx [42] package to implement the graph-based functions of AdaptDQC, including the DQC formulation (Section IV-A), quantitative analysis (Section IV-B) and DQC compilation (Section V).

Platform. In practical scenarios, certain quantum computing platforms are capable of supporting both QubitComm and GateComm, thereby enabling a hybrid communication architecture through AdaptDQC. However, other platforms may be restricted to a singular communication architecture, either QubitComm or GateComm, necessitating AdaptDQC to adapt and utilize a single communication framework correspondingly. To

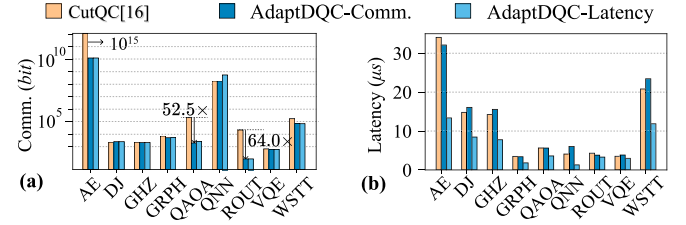


Fig. 8. Performance comparison among QubitComm compilers: (a) comparison in communication cost. (b) Comparison in latency.

address these varied platform capabilities, our research methodology encompasses three distinct experimental groups, each tailored to a specific operational context: 1) Performance under QubitComm architecture, 2) Performance under GateComm architecture, and 3) Performance under hybrid architecture. The experiment results and analysis are in Section VI-B.

B. Performance Analysis

The evaluation primarily centers on two key performance metrics: communication cost and latency. Given that AdaptDQC is an adaptive optimization framework, we categorize its optimization strategies into two types: Communication-Oriented (AdaptDQC-Comm.), and Latency-Oriented (AdaptDQC-Latency).

1) Performance under QubitComm architecture

We first compare the performance metrics of AdaptDQC and CutQC [16] under QubitComm architecture. Fig. 8 presents the results in terms of communication cost (lower is better) and latency (lower is better).

The bar charts in the figure illustrate the average performance values on the circuits of each benchmarking algorithm. We have made the following observations according to the results:

1) *AdaptDQC achieves a significant $15.4 \times$ reduction on communication and 42.5% reduction on latency metrics.* Fig. 8(a) illustrates the performance in terms of communication (Comm.) metrics. We can see that in cases like AE, QAOA, ROUT, AdaptDQC-Comm notably outperforms CutQC. Specifically, the communication cost of AdaptDQC is $52.4 \times$ less for QAOA and about $62.0 \times$ less for ROUT compared to CutQC. This is because despite being communication-optimized, the MIP cutting algorithm that CutQC adopted cannot guarantee optimal solutions and still resorts to heuristic algorithms like branch-and-bound to solve the NPC problem. In contrast, through circuit simplification and graph clustering, AdaptDQC can reduce problem difficulty and has the potential to discover more effective solutions.

Fig. 8(b) compares the performance in terms of the latency metric. Here, AdaptDQC-Latency demonstrates clear advantages against CutQC, achieving an average $1.8 \times$ improvement over CutQC. The achieved latency is consistently below $15 \mu s$. It indicates the latency-oriented clustering algorithm can effectively split the quantum circuit into sub-circuits and minimize the system latency.

2) *AdaptDQC demonstrates good robustness under QubitComm architecture.* In Fig. 8 we also observe that even

though AdaptDQC is optimized toward one of the two metrics, its performance on another metric is still decent. For example, AdaptDQC-Latency is optimized towards latency, but in Fig. 8(a) its communication performance is close to CutQC and even better than CutQC in hard benchmarks like AE, QAOA, and ROUT. AdaptDQC-Comm can also achieve comparable performance in latency against CutQC even though it is optimized toward communication. That is to say, under QubitComm architecture, AdaptDQC is robust and generalizable to performance metrics that are not within the optimization objectives.

2) Performance under GateComm architecture

We then compare AdaptDQC and the baseline approach Hyper [19]+Autocomm [20] under the GateComm architecture. Two observations are made according to the results:

1) *AdaptDQC achieves 26.3% reduction on communication, 18.2% reduction on latency compared to Hyper+Autocomm.* Fig. 9(a) shows the results of the communication metric. Here, AdaptDQC-Comm demonstrates notable advantages against Hyper+Autocomm, reducing an average 26.3% communication cost over Hyper+Autocomm. Especially in circuit cases like AE and QNN, AdaptDQC-Comm reduces over 50% communication cost over baseline. Such an advantage is attributed to the iterative performance evaluation adopted in AdaptDQC. Because AdaptDQC evaluates the communication cost accurately at each iteration of qubit allocation, it can continuously approach the minimum value of the communication cost. On the other hand, the baseline method optimizes the number of remote gates in the circuit during the iteration process, which is not the communication cost in the DQC system, i.e., the number of EPR pairs. Therefore, it cannot effectively find the optimal value.

Fig. 9(b) shows the latency results. From the figure, we can see the AdaptDQC-Latency achieves significant reductions, averaging around 18.2%. Especially in hard cases like AE, QNN, and ROUT, AdaptDQC-Latency reduces the latency by more than 30% compared to Hyper+Autocomm. In relatively simple cases, namely those with lower latency like GRPH, AdaptDQC, and Hyper+Autocomm are almost the same and AdaptDQC's superiority cannot be well demonstrated. Because AdaptDQC directly optimizes for latency as its objective, it inherently possesses advantages over the baseline, which optimizes for indirect metrics as its objective.

2) *AdaptDQC demonstrates good robustness under GateComm architecture.* In Fig. 9 we also observe that even though AdaptDQC is optimized toward one of the two metrics, its performance on another metric is still decent. For example, AdaptDQC-Latency is optimized towards latency, but in Fig. 9(a) its communication performance is still better than Hyper+Autocomm in most benchmarks. Similarly, AdaptDQC-Comm achieves lower latency in all benchmarks against Hyper+Autocomm even though it is optimized toward communication. That is to say, under GateComm architecture, AdaptDQC is also robust and generalizable to performance metrics that are not within the optimization objectives.

3) Performance under hybrid architecture

In this section, we assess the performance of AdaptDQC under hybrid architecture and compare it against AdaptDQC under

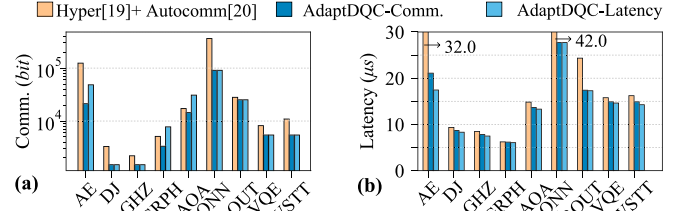


Fig. 9. Performance comparison among GateComm compilers: (a) comparison in communication cost. (b) Comparison in latency.

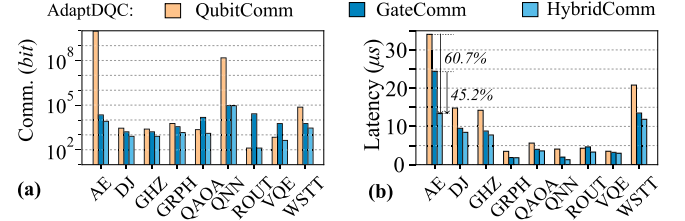


Fig. 10. Experimental performance of QubitComm, GateComm, and HybridComm: (a) comparison in communication cost. (b) Comparison in latency.

QubitComm and AdaptDQC under GateComm, since AdaptDQC has previously demonstrated superior performance over both CutQC [16] and Hyper [19]+AutoComm [20]. Detailed performance analysis is presented as follows: Fig. 10 presents the communication cost (lower is better) and latency (lower is better) of QubitComm, GateComm, and HybridComm. The bar charts in the figure illustrate the average performance values on the circuits of each benchmarking algorithm. We have made the following observations according to the results:

1) *HybridComm achieves a remarkable 35.4% average reduction on communication.* Fig. 10(a) illustrates the performance in terms of communication (Comm.) metrics. In general, HybridComm's communication cost is on average 40.3% less than Gatetcomm and $4181\times$ less than Qubitcomm, especially outperforming in cases like AE, QNN, and WSTT. In the analysis of communication cost, it is evident that GateComm demonstrates a significantly reduced communication cost compared with QubitComm. However, it is noteworthy that in the case of QAOA and ROUT, this trend is dramatically reversed, with QubitComm exhibiting lower communication costs than GateComm. This result reveals that the performance disparities of GateComm and QubitComm are related to quantum algorithms, demonstrating the potential of a hybrid communication framework. AdaptDQC models the two communication in spatial and temporal graph models, thus unlocking the potential for substantial performance improvements.

2) *HybridComm achieves 38.4% average reduction on latency.* Fig. 10(b) compares the performance in terms of the latency metric. Here, HybridComm demonstrates clear advantages against the baselines, achieving an average 48.1% reduction in Latency over QubitComm and a 24.3% reduction over GateComm. The latency of AdaptDQC is consistently below $15\mu s$. In some cases like AE, DJ, and GHZ, the latency is reduced by at least 60% compared with QubitComm. In the

worst case of VQE, there is only a 14.8% reduction. This may be attributed to the inherent simplicity of the circuit structures in these quantum algorithms, precluding significant performance differentiation. Integrating the data presented in Fig. 10(a) and Fig. 10(b), we can observe that in most cases, the GateComm has higher Communication costs and lower Latency compared with QubitComm. This pattern suggests that distinct communication frameworks offer varying performance advantages. As an adaptive algorithm, AdaptDQC can optimize configurations for both communication methods, thereby effectively meeting specific performance criteria required in different scenarios.

3) *HybridComm has better topological adaption on specific hardware.* Fig. 11 presents AdaptDQC's topological optimization on benchmarks. The graph optimization objective of AdaptDQC is set to topological mapping overhead. The topological cost in Fig. 11 is quantified by the swap gate cost when mapping the subcircuits to the hardware. Fig. 11(a) shows the average topology cost of three methods on the benchmarks. We can see that the blue line (representing HybridComm) is always inside the other lines. These results indicate that HybridComm has a lower topology cost than single ICC architectures. For example, Fig. 11(b) shows the topological chip layout of IBM Falcon r4T. The topological mapping results for a QAOA circuit are shown in Fig. 11(c). The first two rows show the subcircuit topology from single architectures, i.e., GateComm and QubitComm, and the last row shows the subcircuit topology of hybridComm. Within these visual representations, blue lines symbolize valid connections in alignment with the chip layout, denoting two-qubit gates that can be directly executed on the chip without necessitating SWAP operations. Conversely, orange lines represent illegal connections that require the introduction of SWAPs to establish viable connections. The number of SWAPs needed for each subfigure is indicated beneath it. Notably, HybridComm necessitates two fewer SWAPs than the single ICC architecture. This efficiency stems from the flexible topology structure of HybridComm with hybrid communication. By leveraging Hybrid communication, HybridComm can generate subcircuits with more flexible topological structures, making it easy to adopt the underlying hardware topology.

C. Real-World Experiments

To demonstrate AdaptDQC's performance on real quantum platforms, we further conduct experiments on IBM's cloud quantum computers. The platform contains a 5-qubit physical machine with a Falcon r4T processor type [43]. Considering that such a platform only contains a single physical quantum chip, to evaluate DQC on this platform, we partition the large circuit into subcircuits, which are deployed on the physical chip in a time-division manner. We use probabilistic reconstruction to establish qubit communication among executed subcircuits, such as by simulating a QubitComm-based DQC system on this platform.

Fig. 12(a) presents a comparison of the chi-square distance between the probability distribution of the output a_i and the noiseless ground truth b_i . The chi-square distance is defined as

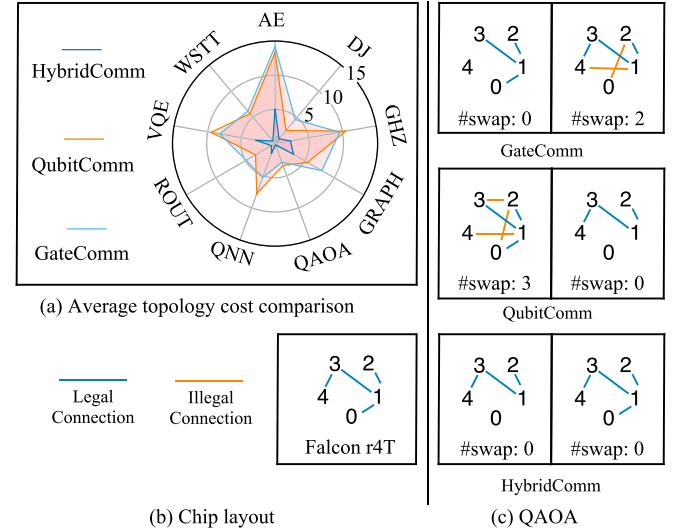


Fig. 11. Topological optimization examples: (a) the average topology cost comparison on different ICC architectures; (b) the physical layout of a 5-qubit IBM Falcon r4T chip; (c) the topological layout results under the QAOA algorithm benchmark.

$\chi^2 = \sum_{i=0}^{2^n-1} \frac{(a_i - b_i)^2}{a_i + b_i}$, which is higher when the two distribution is not identical. The error rate, as quantified on the right y-axis, is calculated by $1 - \text{fidelity}$, and the fidelity is computed by the sum of the subcircuit fidelity on each quantum chip. Since the error of tensor construction is ignorable compared with the quantum noise, we only consider the error of executing the subcircuits on quantum chips. We can see that the error rate and the chi-square distance all increase with the number of qubits. However, the growth trend of the error rate is sublinear to the number of qubits. Especially, on GRAPH algorithms, the error rate increases 0.1% from 7 qubits to 15 qubits but only increases 0.03% from 15 qubits to 21 qubits. This illustrates the error-tolerance property of QubitComm architecture.

Fig. 12(b) displays a comparison between the probability distribution of the outputs obtained from a DQC system (compiled by AdaptDQC) and a noisy single-chip baseline. The single-chip baseline is obtained by executing the original quantum circuit in the IBM Perth chip with seven physical qubits. Two algorithm circuits are considered in the figure: VQE-7, representing a 7-qubit VQE algorithm circuit with discrete peak distribution, and GRAPH-7, representing a Graph State algorithm circuit with a uniform probability distribution. We can observe that when the probability distribution is uniform (i.e., GRAPH-7), AdaptDQC's results are much better than those of the single-chip baseline, showing a distribution (orange line) that closely approximates the ground truth (black line). For probability distributions with multiple concentrated values (i.e., VQE-7), since the noise has a smaller impact on the results, both AdaptDQC and the single-chip baseline achieve satisfactory outputs.

D. Compilation Time

We evaluate the compilation time of AdaptDQC and compare it with the baseline frameworks. We test the end-to-end

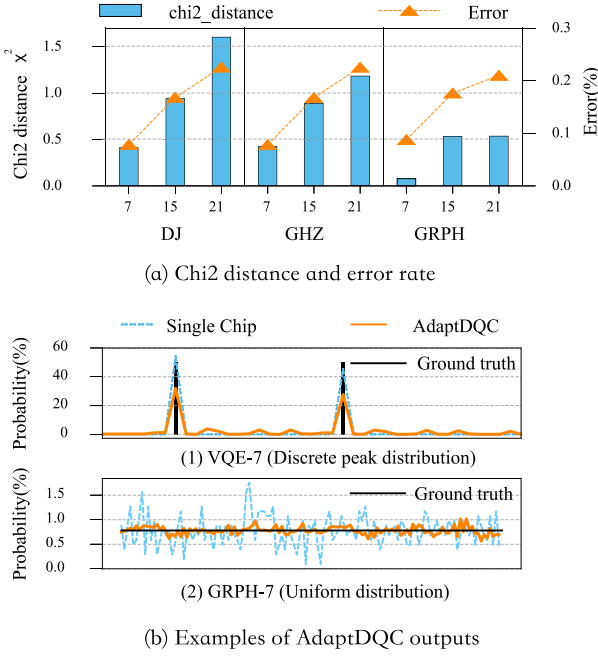


Fig. 12. Real-world comparison between a single chip and AdaptDQC: (a) the chi-square distance between the probability distribution of AdaptDQC and the noise-free ground truth. (b) The output probability distribution obtained by AdaptDQC and a noisy single-chip baseline.

computation elapse time on an AMD Ryzen Threadripper 3960X processor platform.

Fig. 13 depicts the scaling behavior of compilation time for AdaptDQC and CutQC as the quantum circuit size (measured by the number of two-qubit gates) increases. We evaluate the compilation time on different benchmarks and calculate the average time, which is quantified on the y-axis. We can observe from the figure that when the circuit size is small ($\#two\text{-qubit gate} \leq 100$), both algorithms can generate solutions within a reasonable time frame of approximately 10 seconds. However, in such cases, CutQC’s running time is roughly twice that of AdaptDQC. As the quantum circuit size increases, CutQC’s compilation time escalates sharply, following a high-order polynomial trend. In contrast, AdaptDQC continues to exhibit a gradual increase in compilation time, remaining at around 10 seconds. This is because the MIP algorithms in CutQC use heuristic iterative search to find optimal solutions. This makes the solving process unstable and time-consuming. On the other hand, AdaptDQC employs a graph-based clustering algorithm that traverses and partitions spatial and temporal domains. Thus, it has an approximately linear time complexity as the circuit size grows.

Compared to the Hyper+Autocomm baseline for GateComm architectures, AdaptDQC does not show advantages in compilation time. According to our profiling, AdaptDQC is about $16.9\times$ slower than Hyper+Autocomm on average. Such a slowdown is caused by the fact that AdaptDQC adopts an iterative way to optimize the whole system performance, while the baseline only needs two steps for compilation. Though slower than Hyper+Autocomm, we have proven that AdaptDQC can

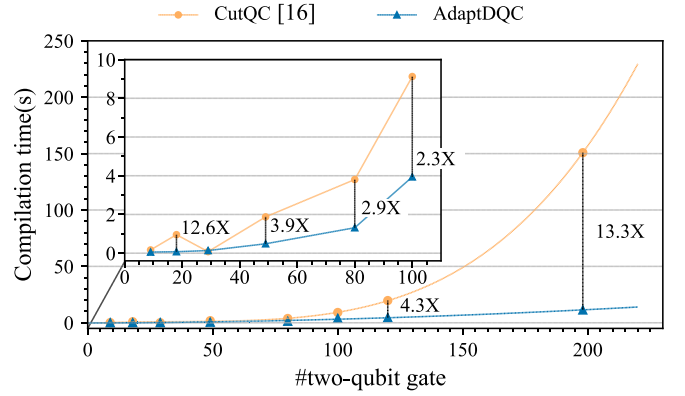


Fig. 13. Compilation time between CutQC [16] and AdaptDQC.

achieve better overall performance than Hyper+Autocomm (see Section VI-B). We can restrict the total rounds of the iteration to strike a balance between compilation time and the DQC performance if necessary. We leave this to future work.

E. Classical Post-Processing Time

We evaluate the classical post-processing time of QubitComm architectures, including AdaptDQC and CutQC [16]. We use the TensorFlow Package for classical tensor computation and test the end-to-end tensor computation time on an AMD Ryzen Threadripper 3960X CPU with 1.5TB memory.

Fig. 14 plots the classical post-processing time of CutQC [16] and AdaptDQC. We average the classical post-processing time of different benchmarks over the circuit size (measured by the number of two-qubit gates, as depicted in the x-axis). From this figure, we can see that the classical post-processing time of AdaptDQC is close to that of CutQC. In some cases, the post-processing time of CutQC is $10.1\times$ at most that of AdaptDQC. Although CutQC aims to find the optimal number of circuit cuts to reduce the classical post-processing overhead, the MIP solver in CutQC cannot ensure optimal results due to heuristic iterative search. AdaptDQC captures the graph structure of quantum circuits, which sometimes can find better solutions than the MIP solver. Because AdaptDQC uses a heuristic graph clustering approach, AdaptDQC has a $6.2\times$ post-processing overhead than CutQC at most. Overall, since finding the optimal number of circuit cuts is an NP-hard problem, both CutQC and AdaptDQC cannot ensure the optimal solution and both have similar post-processing latency.

F. Ablation Study on AdaptDQC Compiler

To analyze the effect of each step in the AdaptDQC Compiler, we evaluate the communication and latency on HybridComm architecture. Since graph simplification does not change the graph structures, we perform an ablation study on the reverse tuning step. The results are shown in Fig. 15. The communication cost and latency data are averaged by the maximum width of QPU, as shown in the x-axis. We can observe that reverse tuning presents a significant reduction in both communication

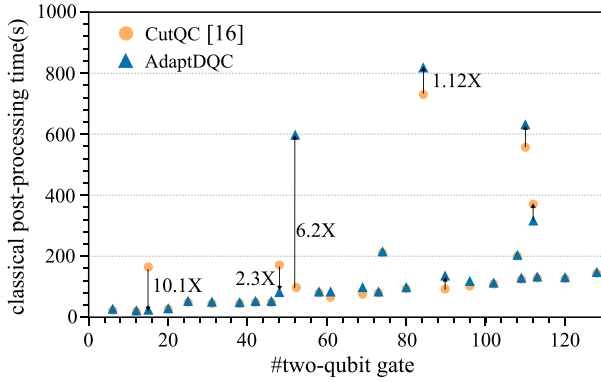


Fig. 14. Classical post-processing time between CutQC [16] and AdaptDQC.

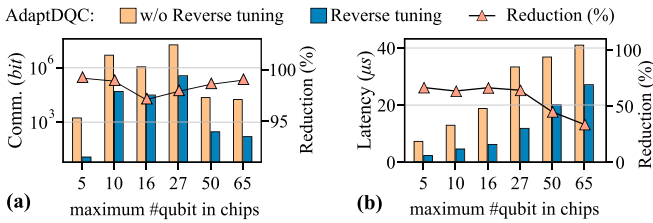


Fig. 15. Ablation study of reverse tuning technique in AdaptDQC compiler.

cost and latency. Specifically, the reverse tuning step reduces 98.5% communication cost and 56.4% latency on average.

VII. RELATED WORK

A. DQC Compiler for Qubit Communication

Currently, the research on qubit communication in Distributed Quantum Computing (DQC) has seen advancements, with Peng et al. [15] proposing a theory that leverages cut qubit wires and classical tensor calculation to achieve parallel computing. The implementation of this theory, known as CutQC [16], utilizes the MIP model to find the minimum cut solution. However, it has been observed that the partitioning algorithm struggles to deliver feasible solutions for large-scale circuits within a reasonable time frame and lacks consideration for optimization directions such as latency.

In this study, we aim to address these limitations by conducting an extensive analysis of various performance indicators, including errors, latency, and communication volume, across the entire system. We present a faster and more universally applicable circuit-cutting scheme achieved through the utilization of a graph clustering optimization algorithm. By employing this approach, we aim to enhance the efficiency of qubit communication in DQC, enabling better performance for large-scale circuits while accounting for optimization considerations like latency.

B. DQC Compiler for Gate Communication

In designing a DQC compiler based on Gate Communication (GateComm) architecture, the current framework separates

qubit distribution and communication optimization. In the qubit distribution scheme, existing frameworks [19], [44] compile circuits into hypergraph or graph and then solve it using graph-cut algorithms. In communication optimization, Autocomm [20] found that the communication effects between qubits and chips in the circuit can be used to reduce teleportation in gate communication. Sundaram et al. [45] combine these works through two compilation steps. However, existing frameworks are generally two-step processes. We combine the two steps together and use the compiled circuit as the minimum target in the first step of qubit distribution, which reduces the EPR requirements. On the other hand, we also analyze the performance indicators of the system based on the spatial-temporal graph model and provide an adaptive compiler.

VIII. CONCLUSION

In this paper, we introduce AdaptDQC, an adaptive compiler framework for optimizing distributed quantum computing under diverse performance metrics and inter-chip communication architectures. Our approach includes formulating DQC using spatial-temporal graphs, quantitative analysis of critical performance metrics, and an adaptive compiler for different optimization objectives. Comprehensive benchmarking demonstrates that AdaptDQC outperforms existing frameworks on various algorithm circuits. Our latency-oriented compiler reduces the average latency by 42.5% in QubitComm architecture and 18.2% in GateComm architecture; our communication-oriented compiler can reduce 15.4 \times and 26.3% communication costs in QubitComm and GateComm, respectively. We demonstrated hybrid ICC architecture can reduce 35.4% communication overhead and 38.4% latency compared with single ICC architectures. The contributions of this work lay a foundation for the future development of high-performance DQC systems.

REFERENCES

- [1] B. Wang, F. Hu, H. Yao, and C. Wang, "Prime factorization algorithm based on parameter optimization of Ising model," *Sci. Rep.*, vol. 10, no. 1, p. 7106, Apr. 2020.
- [2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. 28th Annu. ACM Symp. Theory Comput. (STOC)*, New York, NY, USA: ACM, 1996, pp. 212–219.
- [3] V. Akshay, H. Philathong, M. E. S. Morales, and J. D. Biamonte, "Reachability deficits in quantum approximate optimization," *Phys. Rev. Lett. (PRL)*, vol. 124, p. 090504, Mar. 2020.
- [4] A. Trabesinger, "Quantum simulation," *Nat. Phys.*, vol. 8, no. 4, pp. 153–185, Apr. 2012.
- [5] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, Sep. 2017.
- [6] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," *Theor. Comput. Sci.*, vol. 560, pp. 7–11, Sep. 2014.
- [7] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018.
- [8] B. Lekitsch et al., "Blueprint for a microwave trapped ion quantum computer," *Sci. Adv.*, vol. 3, no. 2, p. e1601540, 2017.
- [9] S. Sheldon, E. Magesan, J. M. Chow, and J. M. Gambetta, "Procedure for systematically tuning up cross-talk in the cross-resonance gate," *Phys. Rev. A (PRA)*, vol. 93, p. 060302, Jun. 2016.
- [10] P. Das, S. Tannu, and M. Qureshi, "Jigsaw: Boosting fidelity of NISQ programs via measurement subsetting," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, New York, NY, USA: ACM, 2021, pp. 937–949.

- [11] D. C. Marinescu and G. M. Marinescu, "Chapter 1 - preliminaries," in *Classical and Quantum Information*, D. C. Marinescu and G. M. Marinescu, Eds. New York, NY, USA: Academic Press, 2012, pp. 1–133.
- [12] H. Buhrman and H. Röhrig, "Distributed Quantum Computing," in *Mathematical Foundations of Computer Science 2003, Ser. Lecture Notes in Computer Science*, B. Rovan and P. Vojtáš, Eds. Berlin, Heidelberg: Springer, 2003, pp. 1–20.
- [13] D. Cuomo, M. Caleffi, and A. S. Cacciapuoti, "Towards a distributed quantum computing ecosystem," *IET Quantum Communication*, vol. 1, no. 1, pp. 3–8, Jul. 2020.
- [14] M. Elkin, H. Klauck, D. Nanongkai, and G. Pandurangan, "Can quantum communication speed up distributed computation?" in *Proc. PODC*, New York, NY, USA: ACM, 2014, pp. 166–175.
- [15] T. Peng, A. W. Harrow, M. Ozols, and X. Wu, "Simulating large quantum circuits on a small quantum computer," *Phys. Rev. Lett.*, vol. 125, no. 15, p. 150504, Oct. 2020.
- [16] W. Tang, T. Tomesh, M. Suchara, J. Larson, and M. Martonosi, "CUTQC: Using small quantum computers for large quantum circuit evaluations," in *Proc. ASPLOS*, New York, NY, USA: ACM, 2021, pp. 473–486.
- [17] M. A. Perlin, Z. H. Saleem, M. Suchara, and J. C. Osborn, "Quantum circuit cutting with maximum-likelihood tomography," *Npj Quantum Inf.*, vol. 7, no. 1, Apr. 2021.
- [18] D. Lago-Rivera, J. V. Rakonjac, S. Grandi, and H. D. Riedmatten, "Long distance multiplexed quantum teleportation from a telecom photon to a solid-state qubit," *Nat. Commun.*, vol. 14, no. 1, p. 1889, Apr. 2023.
- [19] P. Andrés-Martínez and C. Heunen, "Automated distribution of quantum circuits via hypergraph partitioning," *Phys. Rev. A*, vol. 100, no. PRA, p. 032308, Sep. 2019.
- [20] A. Wu, H. Zhang, G. Li, A. Shabani, Y. Xie, and Y. Ding, "AutoComm: A framework for enabling efficient communication in distributed quantum programs," in *Proc. MICRO*, Oct. 2022, pp. 1027–1041.
- [21] C. Hojny, I. Joormann, H. Lüthen, and M. Schmidt, "Mixed-integer programming techniques for the connected max-k-cut problem," *Math. Program. Comput.*, vol. 13, no. 1, pp. 75–132, 2021.
- [22] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. DAC*, 1999, pp. 343–348.
- [23] R. Cane, D. Chandra, S. X. Ng, and L. Hanzo, "Experimental characterization of fault-tolerant circuits in small-scale quantum processors," *IEEE Access*, vol. 9, pp. 162996–163011, 2021.
- [24] K. Fang, J. Zhao, X. Li, Y. Li, and R. Duan, "Quantum NETWORK: From theory to practice," *Sci. China Inf. Sci.*, vol. 66, no. 8, Jul. 2023, Art. no. 180509.
- [25] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?" *Phys. Rev.*, vol. 47, p. 777, May 1935.
- [26] J. Zhao et al., "Enhancing quantum teleportation efficacy with noiseless linear amplification," *Nat. Commun.*, vol. 14, no. 1, 2023, Art. no. 4745.
- [27] H.-J. Briegel, W. Dür, J. I. Cirac, and P. Zoller, "Quantum repeaters: The role of imperfect local operations in quantum communication," *Phys. Rev. Lett.*, vol. 81, pp. 5932–5935, Dec. 1998, doi: 10.1103/PhysRevLett.81.5932.
- [28] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 287–290, 1959.
- [29] M. Suchara, J. Kubiawicz, A. Faruque, F. T. Chong, C.-Y. Lai, and G. Paz, "QURE: The quantum resource estimator toolbox," in *Proc. ICCD*, Piscataway, NJ, USA: IEEE Press, Oct. 2013, pp. 419–426.
- [30] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th Des. Automat. Conf.*, Piscataway, NJ, USA: IEEE Press, 1982, pp. 241–247.
- [31] G. Brassard, P. Høyer, M. Mosca, and A. Tapp, "Quantum amplitude amplification and estimation," pp. 53–74, 2002. [Online]. Available: <http://dx.doi.org/10.1090/conm/305/05215>
- [32] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," in *Proc. Roy. Soc. London. Ser. A: Math. Phys. Sci.*, vol. 439, no. 1907, pp. 553–558, Dec. 1992.
- [33] T. Fritz, "Beyond bell's theorem: Correlation scenarios," *New J. Phys.*, vol. 14, no. 10, Oct. 2012, Art. no. 103001.
- [34] M. Hein, W. Dür, J. Eisert, R. Raussendorf, M. V. den Nest, and H. J. Briegel, "Entanglement in graph states and its applications," *Quantum Phys.*, vol. 162, pp. 115–218, Feb. 2006.
- [35] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, "Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices," *Phys. Rev. X*, vol. 10, no. PRX, p. 021067, Jun. 2020.
- [36] S. C. Kak, *Quantum Neural Computing Adv. Imaging and Electron Physics*, vol. 94, pp. 259–313, May 1995.
- [37] U. Azad, B. K. Behera, E. A. Ahmed, P. K. Panigrahi, and A. Farouk, "Solving vehicle routing problem using quantum approximate optimization algorithm," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 7, pp. 7564–7573, Jul. 2023.
- [38] A. Peruzzo et al., "A variational eigenvalue solver on a photonic quantum processor," *Nat. Commun.*, vol. 5, p. 4213, Jul. 2014.
- [39] A. Cabello, "Bell's theorem with and without inequalities for the three-qubit greenberger-horne-zeilinger and w states," *Phys. Rev. A*, vol. 65, p. 032108, Feb. 2002.
- [40] I. Q. Devices, "IBMQ quantum physical devices," 2022. Accessed: Jun. 20, 2024. [Online]. Available: <https://quantum-computing.ibm.com/services/resources?tab=systems>
- [41] Qiskit Community, "Qiskit: An open-source framework for quantum computing," March 2017. Accessed: May 29, 2024. [Online]. Available: <https://github.com/Qiskit/qiskit>
- [42] M. Treinish, I. Carvalho, G. Tsilimigkounakis, and N. Sá, "RUSTWORKX: A high-performance graph library for Python," *J. Open Source Softw.*, vol. 7, no. 79, p. 3968, Nov. 2022.
- [43] I. Q. Devices, "ibmq quito," 2023. Accessed: Jun. 20, 2024. [Online]. Available: https://quantum-computing.ibm.com/services/resources?tab=systems&system=ibmq_quito
- [44] O. Daei, K. Navi, and M. Zomorodi-Moghadam, "Optimized quantum circuit partitioning," *Int. J. Theor. Phys.*, vol. 59, no. 12, pp. 3804–3820, 2020.
- [45] R. G. Sundaram, H. Gupta, and C. R. Ramakrishnan, "Efficient distribution of quantum circuits," in *Proc. Int. Symp. Distrib. Comput.*, 2021, pp. 41:1–41:20.



Debin Xiang received the B.S. degree in physics from Zhejiang University, in 2024. He is currently working toward the Ph.D. degree with the College of Computer Science, Zhejiang University, Hangzhou. His research interests include quantum computing software and architecture optimization.



related domain, including ICCAD, FPT, HPCC, etc.

Liqiang Lu is an Assistant Professor (ZJU100 Young Professor) with the College of Computer Science, Zhejiang University (ZJU), China. His research interests include quantum computing, computer architecture, deep learning accelerator, and software hardware codesign. He has authored more than 30 scientific publications in premier international journals and conferences in related domains, including ISCA, MICRO, HPCA, ASPLOS, FFCM, DAC, IEEE MICRO, and TCAD. He also serves as a TPC Member in the premier conferences in the



Siwei Tan received the Ph.D. degree in computer science from Zhejiang University, in 2024. He is an Assistant Professor (ZJU100 Young Professor) with the College of Software, Zhejiang University (ZJU), China. His research interests include quantum programming and quantum computer architecture.



Xinghui Jia received the Ph.D. degree from Zhejiang University, Beijing, China, in 2024. His research interest includes quantum computing architecture.



Zhe Zhou received the B.S. degree from Peking University, Beijing, China, in 2019, where he is currently working toward the Ph.D. degree in computer science. His research interests include computer architecture, domain-specific accelerators, and near-memory processing technology.



Mingshuai Chen received the Ph.D. degree in computer science from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 2019. He is a Tenure-Track Assistant Professor leading the Formal Verification Group, Zhejiang University. His research interests include formal verification and synthesis, broadly construed in mathematical logic, and theoretical computer science.



Guangyu Sun (Senior Member, IEEE) received the B.S. and M.S. degrees from Tsinghua University, Beijing, China, in 2003 and 2006, respectively, and the Ph.D. degree in computer science from Pennsylvania State University, State College, PA, USA, in 2011. He is an Associate Professor with the Center for Energy-Efficient Computing and Applications, Peking University, Beijing. His research interests include computer architecture, acceleration systems, and electronic design automation for modern applications. He is currently serving as an Associate

Editor for the *ACM Journal on Emerging Technologies in Computing Systems*. He is a member of ACM and CCF.



Jianwei Yin (Senior Member, IEEE) received the Ph.D. degree in computer science from Zhejiang University (ZJU), in 2001. He was a Visiting Scholar with Georgia Institute of Technology. He is currently a Full Professor with the College of Computer Science, ZJU. He has published more than 100 papers in top international journals and conferences. His research interests include service computing and business process management. He is an Associate Editor of *IEEE TRANSACTIONS ON SERVICES COMPUTING*.