DMapS: End-to-end Qubit Mapping and Routing for Distributed Quantum Computing Architectures

Tingyu Luo, Yuzhen Zheng, Yuxin Deng, and Xiang Fu

***Abstract*—Distributed quantum computing (DQC) architectures offer a scalable solution for the computational demands of large-scale quantum computing. In near-term DQC architectures, the costly remote quantum communication and the execution cost within quantum chips together significantly limit the execution efficiency of quantum circuits. To comprehensively optimize both costs, we propose DMapS, which consists of end-to-end algorithms for qubit mapping and routing. The qubit mapping component, DMapS-M, adopts a two-stage mapping strategy that decomposes a large quantum circuit into smaller ones and parallelizes the qubit mapping on quantum chips. The qubit routing component, DMapS-R, reduces remote quantum communication overhead by prioritizing the insertion of local SWAP gates and further improves transpilation efficiency by exploiting parallelism within chips. Our experimental results show that DMapS-M reduces overall overhead (including both remote quantum communication overhead and local SWAP gate overhead) by an average of 43.44% and 59.72%, respectively, compared to two baseline algorithms, and achieves an average speedup of 87.05x. DMapS-R, compared to the baseline algorithm, reduces overall overhead by an average of 8.85% and achieves an average transpilation speedup of 2.78×. Moreover, compared to the DQC-oriented quantum compiler, DMapS reduces remote communication overhead by an average of 75.16%.**

***Index Terms*—Qubit mapping, qubit routing, distributed quantum computing architecture, remote communication overhead.**

I. Introduction

Quantum computing [1] holds great promise for accelerating solutions to problems such as quantum chemistry simulations [2] and factoring [3]. Despite its potential, the current capabilities of quantum computing are substantially constrained by both the quantity and quality of qubits in a system [4]. Factors such as low fabrication yield rates [5], control crosstalk [6], and other technical challenges make it difficult to integrate a larger number of qubits into individual quantum chips while maintaining high control quality. To address this limitation, distributed quantum computing architectures [7]–[9] have been proposed as a promising solution to scale up quantum computing systems.

This work was supported in part by the National Key R&D Program of China under Grant 2023YFA1009403 and in part by the National Natural Science Foundation of China under Grant 62472175. (*Corresponding authors: Yuxin Deng and Xiang Fu.*)

Tingyu Luo is with Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China (e-mail: listenwetnessluo@163.com).

Yuxin Deng is with Shanghai Key Laboratory of Trustworthy Computing, East China Normal University and MoE Key Laboratory of Interdisciplinary Research of Computation and Economics, Shanghai University of Finance and Economics, Shanghai, China (e-mail: yxdeng@msg.sufe.edu.cn).

Yuzhen Zheng and Xiang Fu are with College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China (e-mail: yuzhen.zheng@quanta.org.cn; xiangfu@quanta.org.cn).

Although two-qubit gates can only be applied to a limited set of physically connected qubit pairs, most quantum algorithms are designed independently of the underlying connection topology. This necessitates quantum compilation [10], particularly the qubit mapping and routing module [11], [12], which maps logical qubits to physical ones and inserts SWAP gates to move logical qubits closer together, ensuring that the required two-qubit gates can be executed. However, existing qubit mapping and routing algorithms struggle to fully address the high cost associated with two-qubit gates between qubits on different chips (remote quantum gates), as well as the accompanying challenges.

In DQC architectures, the implementation of remote quantum gates typically requires consuming Einstein-Podolsky-Rosen (EPR) [13] pairs, whose preparation time can be $\sim$ 12 times of that for local two-qubit gates [14]. If the quantum computing architecture is treated as a large-scale single quantum chip with inter-chip connections regarded as low-quality links, previous qubit mapping and routing algorithms originally targeting individual quantum chips [15]–[19] can be used. However, these methods may introduce many remote quantum gates and reduce the fidelity of the execution result. Alternatively, some researchers have proposed dedicated compilation algorithms for the DQC setting [14], [20]–[26]. These algorithms primarily focus on distributing qubits to different chips and/or routing qubits across chips to reduce remote cost. However, they often overlook the cost of local SWAP gates without providing a routing algorithm for qubit movement inside a chip.

Counterintuitively, our observations reveal that the cost of local SWAP operations is significant in DQC architectures. In our experiments involving benchmarking algorithms of varying sizes, we initially employed two recently proposed DQC-oriented qubit mapping algorithms, namely MHSA [23] and tket_dqc_map (qubit allocation method of tket_dqc [24]) to distribute qubits across chips. Subsequently, the SABRE routing process [16] was utilized to move qubits within individual chips. As illustrated in Fig. 1, the cumulative execution time of local SWAP gates increases rapidly as the number of qubits grows, reaching levels comparable to those of remote quantum gates. However, this issue is largely overlooked by most DQC-oriented quantum compilers. Additionally, large-scale quantum circuits and the various constraints inherent in DQC architectures may lead to substantial execution times for generating efficient compilation results.

Hence, it remains an open challenge how to design efficient end-to-end qubit mapping and routing algorithms that can produce quantum executables targeting DQC architectures with minimal additional overhead. Our observations show that

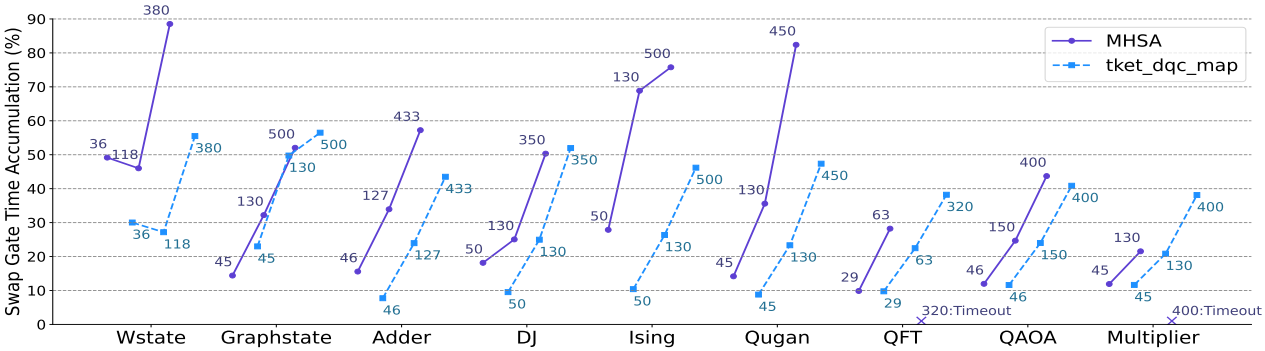


Fig. 1: The ratio of the accumulated execution time of local SWAP gates to the total execution time of both local SWAP gates and remote gates in compiled quantum circuits targeting DQC architectures. The distributed qubit mapping algorithms used are MHSA [23] and tket_dqc_map [24], and the qubit routing method is based on SABRE [16]. The target distributed architectures for three versions of each benchmark with different sizes are shown in Fig. 5 (d) - (f), respectively.

processing the qubit mapping task in stages within DQC architectures maintains mapping quality and boosts algorithm efficiency. We first partition an input quantum circuit by minimizing the number of remote quantum gates, so to reduce the remote EPR pair usage. Then we parallelize qubit mapping within each quantum chip. This two-stage mapping method takes into account both EPR pair usage and local SWAP gate insertions, and accelerates the overall process. For qubit routing, we prioritize inserting local SWAP gates to reduce reliance on remote SWAP gates, thereby lowering remote communication overhead. Moreover, we fully leverage the parallelism of routing tasks within each quantum chip to further improve routing efficiency.

To address this challenge, we propose an end-to-end solution, DMapS, targeting DQC architectures relying on EPR pairs for remote communication. DMapS comprises a qubit mapping algorithm, DMapS-M, and a qubit routing algorithm, DMapS-R. DMapS takes into account both the inter-chip connectivity constraints and the intra-chip topological constraints, which is capable of comprehensively optimizing the overhead associated with EPR pairs and the additional local SWAP gates within individual chips. For DQC architectures employing alternative long-distance communication mechanisms (e.g., long-range couplers [27]), our proposed solution is not specifically tailored, as existing compilation techniques for single quantum chips integrated with long-range couplers can be applied with minor adjustments.

The main contributions of this paper are as follows:

- We are the first to propose end-to-end qubit mapping and routing algorithms for DQC architectures, which jointly optimize EPR pair usage and local SWAP gate insertions.
- We propose a two-stage mapping strategy in DMapS-M, which enables it to reduce both kinds of overhead while ensuring its transpilation efficiency.
- We introduce two primary optimization strategies for DMapS-R. The first prioritizes inserting local over remote SWAP gates. The second leverages the parallelism of routing tasks within quantum chips.
- We extensively evaluate DMapS. ❶ Compared to two state-of-the-art DQC-oriented qubit mapping algorithms, MHSA [23] and tket_dqc_map (qubit allocation method of tket_dqc [24]), DMapS-M achieves significant reductions in EPR pair usage, with an average decrease of 41.35% (up to 98.25%) and 63.35% (up to 99.06%), respectively. In terms of overall overhead (i.e., execution time overhead from remote quantum communication and additional local SWAP gates), DMapS-M achieves an average reduction of 43.44% (up to 90.9%) and 59.72% (up to 96.14%), respectively. Additionally, compared to MHSA, DMapS-M delivers an impressive average transpile speedup of 87.05x (up to 403.33x). ❷ Compared to a SABRE-based qubit routing algorithm [16] improved for DQC architectures, DMapS-R achieves an average reduction in EPR pair usage of 27.26% (up to 84.87%) and an average reduction in overall overhead of 8.85% (up to 46.97%). Furthermore, DMapS-R provides an average transpilation speedup of 2.78x (up to 15.73x). ❸ Compared to the existing DQC-oriented quantum compiler tket_dqc, DMapS reduces EPR pair usage by an average of 75.16% (up to 99.73%).

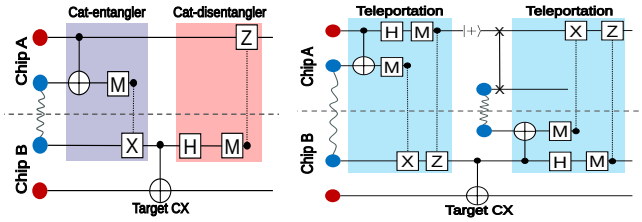
The rest of this paper is organized as follows. Section II outlines the relevant background. Afterward, we introduce the details of DMapS in Section III and evaluate it in Section IV. Finally, Section V reviews related works and Section VI concludes this paper.

II. Background

A. Quantum Compilation

Due to limitations of NISQ devices, such as restricted topology connectivity and native physical gates, high-level quantum programs cannot be directly executed. For instance, two-qubit gates can only be performed between adjacent physical qubits. Therefore, movement operations like SWAP gates must be inserted to enable the execution of quantum gates between non-adjacent physical qubits. To bridge the gap between quantum programs and NISQ devices, various quantum compilation techniques, such as gate decomposition [11], and qubit mapping and routing [15]–[19], have been proposed.

In quantum program compilation, qubit mapping and routing are two crucial steps. (i) **Qubit mapping** generates the optimal initial qubit placements that can achieve some goals such as minimizing the number of additional SWAP gates [18]. (ii) **Qubit routing** aims to continuously insert movement operations such as SWAP gates into the input



(a) *Cat-Comm* Pattern (b) *TP-Comm* Pattern

Fig. 2: Details of two remote quantum communication patterns. (a). *Cat-Comm*, implemented using cat-entangler and cat-disentangler. (b). *TP-Comm*, implemented using quantum teleportation.

quantum programs to relocate the logical qubits that are mapped onto non-adjacent physical qubits [16].

B. Distributed Quantum Computing

Currently, prototype DQC architectures use specially designed superconducting coaxial cables [7] as remote physical connections. These enable physical qubits on different quantum chips to be entangled into EPR pairs, such as the Bell state $(\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle))$, forming remote quantum communication channels in DQC. Current DQC architectures are still in their infancy, facing challenges like the vulnerability of entangled states to noise [28] and the extended duration required for executing remote quantum communications [14].

As shown in Fig. 2, there are two main remote quantum communication patterns for implementing remote quantum gates, known as *Cat-Comm* and *TP-Comm* [14]. A *Cat-Comm* pattern is constructed using a cat-entangler and a cat-disentangler [1], while a *TP-Comm* pattern relies on quantum teleportation [29] to transmit quantum information. The number of EPR pairs consumed in executing remote quantum gates over a DQC architecture is influenced by various factors, including the remote communication pattern used, the distance between the quantum chips hosting the qubits, and the measurement success rate affected by noise, etc. Specifically, executing a remote CNOT gate on two adjacent quantum chips using the *Cat-Comm* pattern consumes one EPR pair, and executing a remote SWAP gate may require up to three EPR pairs [25]. In contrast, using the *TP-Comm* pattern to perform any remote two-qubit gate on two adjacent quantum chips requires the consumption of two EPR pairs.

The term *communication qubits* (as shown by the red dots in Fig. 2) is used to describe physical qubits that have the ability to create remote EPR pairs. Conversely, the physical qubits that serve the purpose of storing program information are termed *data qubits* (as shown by the blue dots in Fig. 2). In the near term, the quantity of communication qubits available on individual quantum chips is limited.

C. Hypergraph and Hypergraph Partitioning

A hypergraph $H = (X, E)$ is a graph with a set of vertices X and a set of hyperedges E, where a hyperedge $e \in E$ is defined as a non-empty set of vertices $e \subset X$. A classical graph only allows an edge to connect two vertices, while a hyperedge can connect more than two vertices. Both vertices and hyperedges can be weighted. k-way hypergraph partitioning is to assign vertices of X to k disjoint non-empty partitions $X_1, X_2, \cdots, X_k$, where $X_i \subset X$, $X_i \cap X_j = \emptyset$ for $i \neq j$, and $\bigcup_i X_i = X$. During partitioning, it normally seeks to minimize some cost function, such as the number or the total weight of hyperedges that span more than one partition.

III. Qubit Mapping and Routing for DQC

This section introduces the proposed qubit mapping and routing algorithm (DMapS), which comprises three steps: preprocessing (Section III-A), the two-stage mapping strategy, DMapS-M (Section III-B), and the routing algorithm tailored for DQC architectures, DMapS-R (Section III-C). Section III-D concludes with the complexities of both algorithms.

A. Preprocessing

In order to perform DMapS targeting a particular DQC architecture, we need to know how far two chips (qubits) are, w.r.t. the minimum number of hops required for information transmitted from one chip (qubit) to the other. We use two distance matrices, D and M, to represent the distance between chips and qubits, respectively. The preprocessing stage can calculate D and M using some All-Pairs Shortest Path (APSP) algorithm, like Floyd-Warshall [30]. Since M is not required for estimating the EPR pair overhead in subsequent usage, the weight of inter-chip remote connections can be set to 1, the same as the weight for intra-chip connections.

B. DMapS-M: Qubit Mapping Targeting DQC Architectures

DMapS-M is an end-to-end qubit mapping algorithm that maps logical qubits to physical ones in DQC architectures. It adopts a two-stage mapping strategy to minimize both the number of EPR pairs and inserted local SWAP gates required to accomplish the following qubit routing process. The workflow of DMapS-M is shown in Fig. 3.

Since remote quantum communication greatly affects the execution efficiency of the final result, reducing remote two-qubit gates is essential. A natural choice is to first group logical qubits with more interconnections in the program and then map logical qubit groups to different quantum chips (inter-chip qubit mapping). Thereafter, intra-chip qubit mapping can be performed, which allocates every logical qubit to a concrete physical qubit. In this way, the complexity of qubit mapping can be reduced in a remarkable way.

*1) **Mapping Qubits to Quantum Chips**:* Mapping qubits to chips can be done in three steps: (1) quantum chip combination selection, (2) qubit grouping, and (3) qubit group allocation.

(1) Quantum chip combination selection. There can be various combinations of chips that can provide enough qubits for the target program. Since using more chips leads to more distributed qubit allocation, which in turn increases the EPR pair consumption, DMapS-M only considers using compact combinations ($\mathcal{S} = \{S_1, S_2, \cdots, S_m\}$) of chips to execute the program. By "compact", it means the chip combination S_i can only provide enough ($\geq n$, where n is the width of quantum circuits) qubits with all chips within S_i.

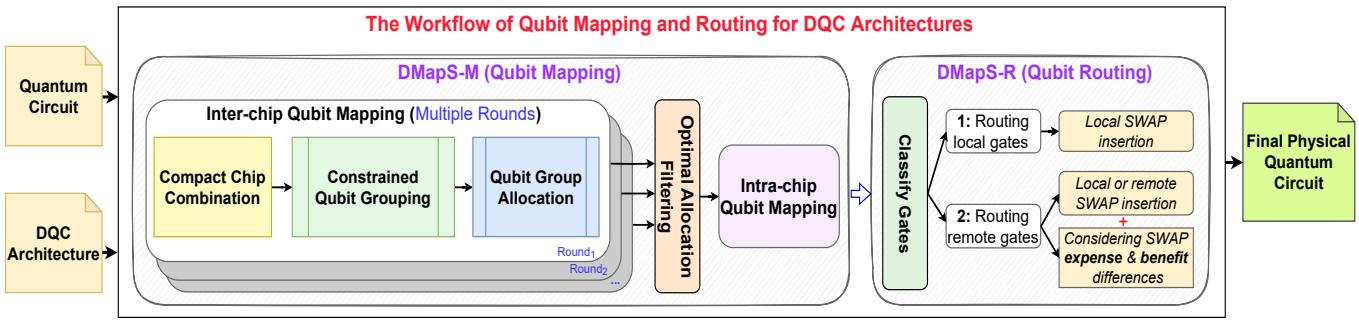


Fig. 3: The workflow of DMapS

(2) Qubit grouping. Previous works [20], [22] have shown that the result of qubit grouping effectively affects the number of remote gates. Recall the definition of hypergraph (Section II-C), if we let a vertex denote a qubit, a hyperedge denote a set of qubits, it is possible to formalize the qubit grouping problem as the weighted hypergraph partitioning problem.

To be more concrete, the hyperedge is defined in the following way so that the partitioning objective function (total weight of hyperedges spanning multiple partitions) can roughly represent the number of required remote gates. To share EPR pairs for multiple remote gates, both *Cat-Comm* and *TP-comm* patterns require these two-qubit gates to be consecutive with regard to a qubit. Hence, we define a **two-qubit gate block** as a list of consecutive two-qubit gate applied on the same qubit, while disregarding any intervening single-qubit gates. The set of qubits involved in such a block is treated as a hyperedge. For instance, consider the sequence $\{tg_2(q_0,q_1), tg_2(q_0,q_2), tg_2(q_0,q_3), tg_2(q_3,q_4)\}$. We identify consecutive two-qubit gates involving qubits ranging from q_0 to q_4. The first three gates form a block due to their common involvement of q_0, resulting in the hyperedge (q_0,q_1,q_2,q_3). The last gate forms a separate block, corresponding to the hyperedge (q_3,q_4).

The set of two-qubit blocks for the same quantum circuit is not unique, which depends on the way to generate them. One way to generate two-qubit blocks is to walk through the two-qubit gate list for every qubit iteratively. During walking over qubit q, assume a two-qubit gate tg_1 is already in a block b, the next two-qubit gate $tg_2(q,q')$ can be added to b if and only if the the other predecessor of tg_2 has been added to some block. We use "the other predecessor" to indicate the two-qubit gate applied on q' just before tg_2. By repeating the above process multiple times, all two-qubit gates can be included in a two-qubit block, or a hyperedge. Then, the weight of a hyperedge can be set as the number of two-qubit blocks on the corresponding qubit set.

Thereafter, we can input the constructed hypergraph with a list of maximum group sizes as constraints to a third-party tool, e.g., KaHyPar [31], to solve the constrained hypergraph partitioning problem and obtain the candidate qubit groups. At this point, due to inter-chip connectivity constraints, the allocation between qubit groups and quantum chips still offers significant optimization potential, and their mapping remains undefined. Moreover, group size constraints are primarily used to ensure feasible qubit groups for subsequent steps, which differs notably from the objective of [20].

Note that for every valid chip combination $S_i \in \mathcal{S}$, qubit grouping should be performed once and get a corresponding grouping result $G_i \in \mathcal{G}$.

(3) Qubit group allocation. For a given quantum chip combination $S_a = \{s_{a,1}, s_{a,2}, \cdots, s_{a,k}\}$ and qubit groups $G_a = \{g_{a,1}, g_{a,2}, \cdots, g_{a,k}\}$, qubit group allocation aims to find a mapping $\omega : G_a \rightarrow S_a$ while satisfying the qubit capacity limitation of every chip. The optimization target of qubit group allocation is to minimize the overhead introduced by remote quantum communication, including the overhead of remote gates and local SWAP gates used to connect two communication qubits on the same chip.

Let $s'_i = \omega(g_{a,i})$ denote the mapped chip for qubit group g_a under the mapping ω. The overhead is then formulated as:

$$\min_{\omega} \sum_{g_{a,i}} \sum_{g_{a,j}} x_{ij} R_{ij} \times \left(\mathcal{W} \cdot e_{s'_i,s'_j} + L(s'_i, s'_j) \right) \tag{1}$$

where R_{ij} is the number of remote gates between $g_{a,i}$ and $g_{a,j}$. Moreover, $x_{ij} = 1$ iff both $g_{a,i}$ and $g_{a,j}$ can be successfully allocated to s'_i and s'_j, respectively, otherwise, an unsuccessful allocation means $x_{ij} = 0$.

The left part in the brace represents the remote gate overhead, where $e_{s'_i,s'_j}$ denotes the EPR pair count potentially used for one remote gate between s'_i and s'_j, and $\mathcal{W}$ is a tunable factor representing the relative weight of remote gates compared to local SWAP. $e_{s'_i,s'_j}$ can be calculated using:

$$e_{s'_i,s'_j} = 2D[s'_i][s'_j] - 1 \tag{2}$$

where $D[s'_i][s'_j]$ is the shortest distance between s'_i and s'_j (see Section III-A).

When applying a remote gate on two qubits on different chips (s_1, s_2), local SWAP gates might also be required to connect the two communication qubits (e.g., $q_{c,1}, q_{c,2}$) on the bridging quantum chips (e.g., s_3), where q_{c_1} (q_{c_2}) connects to s_1 (s_2). $L(s'_i, s'_j)$ represents such local overhead in all bridging chips introduced by a remote gate between s'_i and s'_j.

In classical distributed computing, the task allocation problem, which minimizes the total global communication overhead under specific constraints, are proven to be NP-hard [32]. The qubit group allocation problem shares the problem structure. Hence, it is NP-hard and there is no polynomial-time algorithm to find the optimal result. In this work, we employ the tabu search heuristic algorithm [33] to efficiently find a mapping between qubit groups and quantum chips.

After qubit group allocation, the overhead for all quantum chip combinations is estimated. The best combination and

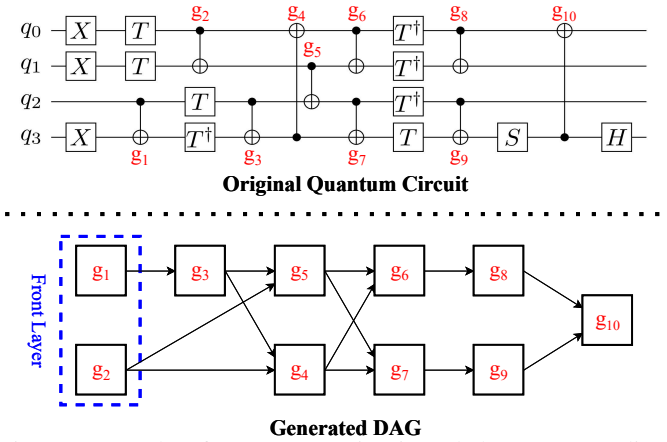


Fig. 4: Example of a quantum circuit and the corresponding DAG generated from it.

corresponding qubit group allocation result are then selected as input for the next stage: intra-chip qubit mapping.

*2) **Intra-chip Qubit Mapping**:* In the second stage, once qubit groups are assigned to quantum chips, qubit-level mapping involves the concurrent mapping of logical qubits from each qubit group to the physical qubits of the respective quantum chip. To accelerate this process, we apply two strategies.

The first strategy performs qubit mapping in parallel within each chip. However, since those two logical qubits affected by remote gates belong to different qubit groups allocated to different quantum chips, the intra-chip qubit mapping tasks are no longer inherently independent. As usual the objective function of a qubit mapping problem typically represents the estimated overhead introduced in the subsequent qubit routing [16]. We observe that the routing cost of remote gates consists of two parts: (i) local SWAP gate insertions within quantum chips and (ii) the remote communication cost between quantum chips. Once the binding relationship between the qubit groups and the quantum chips is determined, the computation of (ii) is fixed compared to that of (i). Based on this observation, the estimated routing costs for remote gates can be simplified to the routing costs within the quantum chips, allowing these tasks to be handled independently.

The second strategy aims to efficiently generate qubit mappings for each qubit group. Inspired by SABRE [16], we perform multiple forward and backward scans on the subcircuits obtained from the partitioning process. The time complexity of this heuristic method depends on the number of two-qubit gates and physical qubits. Its efficiency comes from: (i) Each subcircuit contains fewer two-qubit gates than the entire quantum circuit. (ii) The processing overhead per gate is reduced. This heuristic strategy has a time complexity of $O(m^{2.5})$ per two-qubit gate, where m is the number of physical qubits in each quantum chip, which is significantly smaller than the scale of the entire DQC architecture.

When scanning directed acyclic graphs (DAG) (Fig. 4) of subcircuits, the second strategy maintains two key data structures: the front layer (denoted as F) and the extended set (denoted as E). F contains all two-qubit gates without unexecuted predecessors in the DAG, while E includes gates from deeper levels. Furthermore, a heuristic function H based on nearest neighbor cost (the minimal number of SWAP gates needed to bring two logical qubits adjacent on the quantum chip) is used to select the optimal SWAP to adjust the mapping situation. The details are as follows:

TABLE I: Classification of non-executable CNOT gates.

Gate	Definition
g_l	The qubits affected by this **local** CNOT gate are mapped to the same quantum chip
g_{rao}	The qubits affected by this **remote** CNOT gate are mapped to **adjacent** quantum chips, where **one** is mapped to a data qubit connected to a communication qubit
g_{rnt}	The qubits affected by this **remote** CNOT gate are mapped to **non-adjacent** quantum chips, where **both** are mapped to data qubits connected to communication qubits
g_{ran}	This CNOT gate is **remote**, with two possible cases: (i) the affected qubits are mapped to **adjacent** quantum chips, but **neither** is mapped to a data qubit connected to a communication qubit. (ii) the affected qubits are mapped to **non-adjacent** quantum chips, with **at most** one mapped to such a data qubit.

$$
\begin{aligned}
H = \frac{1}{|F|} \times \{ & \sum_{g_l \in F} M[\pi(g_l.q_1)][\pi(g_l.q_2)] + \\
& \sum_{g_r \in F} (M[\pi(g_r.q_l)][P(g_r.q_r)] + 1)\} + \\
& \frac{W_1}{|E|} \times \{ \sum_{g_l \in E} M[\pi(g_l.q_1)][\pi(g_l.q_2)] + \\
& \sum_{g_r \in E} (M[\pi(g_r.q_l)][P(g_r.q_r)] + 1)\}
\end{aligned}
\tag{3}
$$

where M represents the distance matrix between physical qubits (see Section III-A), and π denotes the qubit mapping relationship. g_l denotes a local two-qubit gate acting on logical qubits q_1 and q_2, while g_r represents a remote two-qubit gate acting on logical qubits q_l (located on the current chip) and q_r (located on another quantum chip). The associated physical data qubit, denoted as $P(g_r.q_r)$, resides on the current chip and connects to $P(g_r.q_r)'$, which is one of the physical communication qubits that are at the shortest distance between the quantum chip where q_l is mapped and the quantum chip where q_r is mapped. The weight parameter W_1, $0 \leq W_1 < 1$, is used to diminish the effect of the second term.

C. DMapS-R: Qubit Routing for DQC Architectures

In the presence of both inter-chip connectivity and intra-chip topological constraints, the end-to-end algorithm DMapS-R aims to efficiently route both remote and local quantum gates at the physical qubit level.

*1) **The workflow of DMapS-R**:* DMapS-R deals with the CNOT gates in the front layer F by iteratively inserting SWAP gates, thereby accomplishing a comprehensive scan of the quantum circuit DAG.

As shown in Fig. 3 and Algorithm 1, two SWAP gates insertion strategies are adopted to compress EPR pair overhead. (i) The first prioritizes local SWAP gate insertion to replace remote ones, thereby avoiding unnecessary EPR pair overhead. Moreover, this strategy may lead to an excessive number of SWAP gates, as DMapS-R leverages multiple heuristic cost functions (i.e., H_{ll} and H_{rl}) for optimization. (ii) The second strategy involves considering the expense and benefit

Algorithm 1: Heuristic Search of DMapS-R

Input : Initial Front layer F, Mapping π, Physical Qubit Affiliation σ, Circuit DAG, Chip Distance Matrix D, Qubit Distance Matrix M
Output: Inserted SWAPs, Final Mapping π_f

1 **while** $F \neq \emptyset$ **do**
2 **if** F *contains executable* CNOT *gates* **then**
3 Remove the executable gates from F and add their successor gates from DAG to F;
4 Continue;
5 **end**
6 Classify the CNOT gates in F into four types: G_l, G_{rao}, G_{rnt}, and G_{ran}, and derive related extended set: E_l, E_{rao}, E_{rnt}, and E_{ran};
7 **if** $G_l \neq \emptyset$ ***or*** *(*$G_{ran} \neq \emptyset$ ***and*** *all other* G *are empty)* **then**
8 **if** $G_l \neq \emptyset$ **then**
9 SWAPs $\leftarrow$ Find optional local SWAP gates;
10 **end**
11 **else**
12 $(FSs, ESs) \leftarrow Each\_Chip\_Layer(G_{ran}, E_{ran}, \sigma, \pi)$;
13 **for** $(FE, ES) \in (FEs, ESs)$ **do**
14 SWAPs $\leftarrow$ Find optional local SWAP gates within the corresponding chip;
15 Find the SWAP with minimal score by evaluating the cost function $H_{rl}(FS, ES, \pi, M)$;
16 $\pi = \pi.update(\text{SWAP})$;
17 **end**
18 Continue;
19 **end**
20 **end**
21 **else if** $G_l = \emptyset$ ***and*** $G_{rnt} \neq \emptyset$ **then**
22 SWAPs $\leftarrow$ Find optional remote SWAP gates;
23 **end**
24 **else if** $G_l = \emptyset$ ***and*** $G_{rnt} = \emptyset$ ***and*** $G_{rao} \neq \emptyset$ **then**
25 SWAPs $\leftarrow$ Find optional local and remote SWAP gates;
26 **end**
27 Select the SWAP with minimal score in SWAPs by evaluating the corresponding cost function: $H_{ll}(G_l, E_l, \pi, M)$, $H_{rr}(G_{rnt}, E_{rnt}, \sigma, \pi, D, M)$, or $H_{rrl}(G_{rao}, E_{rao}, \pi, D, M)$;
28 $\pi = \pi.update(\text{SWAP})$;
29 **end**

differences in executing local and remote SWAP gates. In the scenarios where both local and remote SWAP gates may need to be inserted before proceeding to the subsequent routing, this strategy seeks to maximize the optimization of inter-chip and intra-chip overhead caused by their selection.

During DAG scanning, the non-directly executable CNOT gates in front layer F are classified into four types (see Table I, line 6 in Algorithm 1). A non-directly executable CNOT gate refers to one whose logical qubits q_i and q_j are not mapped to physically connected qubits. To ensure efficient qubit routing, DMapS-R restricts its search to candidate SWAP gates associated with the qubits in G_l, G_{rnt}, G_{rao}, and FS. Here, G_l, G_{rnt}, and G_{rao} refer to the gate sets corresponding to g_l, g_{rnt}, and g_{rao}, respectively, while FS represents the front layer of the corresponding subcircuit.

Suppose q_i is a target of a CNOT gate in G_l, DMapS-R first identifies its mapped physical qubit $Q_i = \pi(q_i)$ (where π denotes the qubit mapping relationship) and all its neighbors $Q_{i1}, \ldots, Q_{i4}$. It then applies the reverse mapping to determine the corresponding logical qubits $q_{i1}, \ldots, q_{i4} = \pi^{-1}(Q_{i1}), \ldots, \pi^{-1}(Q_{i4})$. For logical qubit pairs $(q_i, q_{i1}), \ldots, (q_i, q_{i4})$, the corresponding type of SWAP gates on these qubit pairs are added to SWAPs. This procedure is repeated for all qubits in G_l. The SWAPs for other gate sets (e.g., G_{rnt}, G_{rao}) are generated similarly (see lines 9, 14, 22, and 25 in Algorithm 1). Additionally, various extended sets (E_l, E_{rao}, E_{rnt}, and E_{ran}) are utilized to enable look-ahead ability, with their sizes directly correlated to those of G_l, G_{rao}, G_{rnt}, and G_{ran}, respectively.

Furthermore, DMapS-R performs parallel insertion of local SWAP gates within the corresponding quantum chip only when the front layer consists solely of g_{ran} gates (see Algorithm 1, lines 13–17), thereby enhancing routing efficiency.

*2) **Cost function design**:* Four different heuristic cost functions named H_{ll}, H_{rr}, H_{rrl}, and H_{rl} are designed to evaluate each different type of candidate SWAP (as shown in lines 15 and 27 of Algorithm 1).

(1) Heuristic cost function for rating local SWAP **candidates.** When G_l is not empty, we employ the heuristic cost function H_{ll} in (4) to assess all potential local SWAP candidates. The weight parameter W_2, with $0 \leq W_2 < 1$, is designed to mitigate the influence of the second term.

$$H_{ll} = \frac{1}{|G_l|} \times \sum_{g \in G_l} M[\pi(g.q_1)][\pi(g.q_2)] + \frac{W_2}{|E_l|} \times \sum_{g \in E_l} M[\pi(g.q_1)][\pi(g.q_2)] \tag{4}$$

When only G_{ran} is not empty, we also focus solely on inserting local SWAP gates. Based on the qubits in G_{ran}, the current qubit placement π, and the physical qubit affiliation σ (indicating which chip each physical qubit belongs to), DMapS-R deduces the corresponding quantum chips where local SWAP gates can be inserted in parallel. Then it invokes the function $Each\_Chip\_Layer()$ to extract, for each quantum chip, the independent set FS and extended set ES from G_{ran} and E_{ran}, respectively (as shown in the line 12 of Algorithm 1). The cost function H_{rl} is defined in (5).

$$H_{rl} = \frac{1}{|FS|} \times \sum_{g \in FS} min\_dist(g, \pi, M) + \frac{W_2}{|ES|} \times \{ \sum_{g \in ES_l} M[\pi(g.q_1)][\pi(g.q_2)] + \sum_{g \in ES_r} min\_dist(g, \pi, M) \} \tag{5}$$

In Equation (5), the sets ES_l and ES_r refer to the local CNOT gate set and the remote CNOT gate set within ES. The definition of W_2 aligns with the definition provided earlier. In each corresponding quantum chip, min_dist denotes the minimum value of the shortest distance between a physical data qubit not participating in remote communication and the set of physical data qubits that are actively involved in such communication. The definition of min_dist is as follows:

$$min\{M[\pi(g.q_1)][CQ_1], \ldots, M[\pi(g.q_1)][CQ_k]\} \quad (6)$$

where the physical qubit $\pi(g.q_1)$ that the logical qubit q_1 is mapped does not engage in remote communication. Assume that the physical qubit $\pi(g.q_1)$ resides on the quantum chip c_i, and the other logical qubit q_2 acted upon by gate g is allocated to the quantum chip c_j. Then, CQ denotes the data qubits that are connected to the communication qubits closest to the quantum chip c_j, and k represents the count of the aforementioned data qubits.

(2) Cost function for rating remote SWAP **candidates.** The cost function H_{rr} in (7) is utilized to rate all potential remote SWAP candidates.

$$\begin{aligned} H_{rr} = & \frac{1}{|G_{rnt}|} \times \{ \sum_{g \in G_{rnt}} M[\pi(g.q_1)][\pi(g.q_2)] + \\ & \mathcal{W}_r \times \sum_{g \in G_{rnt}} f(D[\sigma(\pi(g.q_1))][\sigma(\pi(g.q_2))]) \} + \\ & \frac{W_3}{|E_{rnt}|} \times \{ \sum_{g \in E_{rnt}} M[\pi(g.q_1)][\pi(g.q_2)] + \\ & \mathcal{W}_r \times \sum_{g \in E_{rnt}} f(D[\sigma(\pi(g.q_1))][\sigma(\pi(g.q_2))]) \} \end{aligned} \quad (7)$$

In Equation (7), the function f estimates the anticipated EPR pairs usage for executing remote CNOT gates. Parameter $\mathcal{W}_r$ represents the weight ratio between the remote communication overhead and the cost of executing local SWAP gates. The weight parameters W_3, with $0 \leq W_3 < 1$, is used to mitigate the influence of related extended set. Let c_i and c_j denote the quantum chips $\sigma(\pi(g.q_1))$ and $\sigma(\pi(g.q_2))$, respectively. The symbol σ represents the affiliation between the physical qubits and the quantum chip they are located in. The function $f(D[\sigma(\pi(g.q_1))][\sigma(\pi(g.q_2))])$ is defined as:

$$f(D[c_i][c_j]) = 2D[c_i][c_j] - 1 \quad (8)$$

where $D[c_i][c_j]$ denotes the number of routing hops between quantum chips c_i and c_j.

(3) Cost function for rating local and remote SWAP **candidates.** The cost function H_{rrl} in (9) is employed to evaluate all local and remote SWAP candidates.

$$\begin{aligned} H_{rrl} = & Cost(\text{SWAP}) + \\ & \{ \sum_{g \in G_{rao}} M[\pi(g.q_1)][\pi(g.q_2)] + \\ & \mathcal{W}_r \times \sum_{g \in G_{rao}} f(D[\sigma(\pi(g.q_1))][\sigma(\pi(g.q_2))]) \} + \\ & W_3 \times \{ \sum_{g \in E_{rao}} M[\pi(g.q_1)][\pi(g.q_2)] + \\ & \mathcal{W}_r \times \sum_{g \in E_{rao}} f(D[\sigma(\pi(g.q_1))][\sigma(\pi(g.q_2))]) \} \end{aligned} \quad (9)$$

In Equation (9), $Cost(\text{SWAP})$ denotes the estimated operational overhead of SWAP candidates, reflecting the cost disparity between local and remote SWAP gates. This cost function accounts for the different impact of using *Cat-Comm* and *TP-Comm* on the EPR pair overhead for quantum gates in both the front layer and the extended set. The definitions of $\mathcal{W}_r$, W_3, f, σ, and π follow prior usage.

D. Complexities Analysis

We assume the input quantum circuit has n logical qubits and t two-qubit gates, requiring k chips. In the DQC architecture, the total number of physical data qubits is M, with each uniform quantum chip containing m physical data qubits.

For DMapS-M, the first-stage mapping comprises constrained circuit partitioning and qubit group allocation, with time complexities of $O(\frac{\alpha t n^4}{k^4})$ (see KaHyPar analysis in [34]) and $O(\alpha i_1 k^4)$, respectively, where α denotes the number of rounds and i_1 is the number of allocation search iterations. During the second stage mapping process, since intra-chip qubit mapping processes are parallelized, the time complexity of this part is $O(i_2 t m^{2.5})$, where i_2 represents the number of search iterations in these processes. The space complexity of this part is $O(km^3)$. In summary, the time complexity of DMapS-M is $O(\frac{\alpha t n^4}{k^4})$, and the space complexity is $O(km^3)$.

For DMapS-R, the most time-consuming part is the process of inserting SWAP gates. For each two-qubit gate, at most $O(\sqrt{N})$ SWAP gates may be required, corresponding to the diameter of DQC architecture (where N is the total number of physical qubits). Consequently, for each two-qubit gate, the time complexity of the SWAP-based search scheme is $O(N^{2.5})$. Thus, the overall time complexity of DMapS-R is $O(tN^{2.5})$, and its space complexity is $O(N^2)$.

IV. Evaluation

To validate our proposed qubit mapping and routing algorithm for DQC, we conduct a series of simulation experiments. The proposed algorithms and evaluation benchmarks are available at https://github.com/RoccoLoter/DMapS.

A. Simulation Setting

Benchmarks: The 27 benchmark programs used in this study are selected from QASMBench [35] and MQT-Bench [36], covering a wide range of application domains (e.g., quantum arithmetic, quantum machine learning, etc.) and circuit scales ranging from 29 to 500 qubits. They are manually classified into three categories: small-scale (29 to 50 qubits), medium-scale (63 to 150 qubits), and large-scale (320 to 500 qubits). Detailed information is provided in Table II.

DQC architectures: We employ the coupling graphs of three superconducting quantum chips (cf. Fig. 5), namely *Zuchongzhi_2.1* (66 qubits) [37], *IBMQ_Mumbai* (27 qubits) [38], and *IBMQ_Washington* (127 qubits) [39], to construct three DQC architectures for simulation. (i) ***Net_Zuchongzhi*** (Fig. 5d): It is composed of a *Zuchongzhi_2.1* divided into five regions, with each region representing a node. A red double line between two nodes indicates

TABLE II: Benchmark programs

Type	Name	#Qubits	#Gates	#CX
Logical Operation	W-state Preparation and Assessment (Wstate)	36 118 380	281 937 3033	70 234 758
	Graph State (Graphstate)	45 130 500	90 260 1000	45 130 500
Quantum Arithmetic	Quantum Ripple-carry Adder (Adder)	46 127 433	706 1975 7921	325 910 3120
	Quantum Multiplier (Multipler)	45 130 400	12785 52192 458512	2574 22464 190720
Hidden Subgroup	Deutsch-Jozsa Algorithm (DJ)	50 130 350	222 388 1048	49 129 349
	Quantum Fourier Transform (QFT)	29 63 320	2059 8689 60860	812 3400 23960
Quantum Simulation	Ising Model Simulation (Ising)	50 130 500	544 1424 5494	98 258 998
Quantum Machine Learning	Generative Adversarial Network (Qugan)	45 130 450	2066 6180 21700	344 1020 3580
Search and Optimization	QAOA	46 150 400	4416 40650 152400	1380 9000 48000

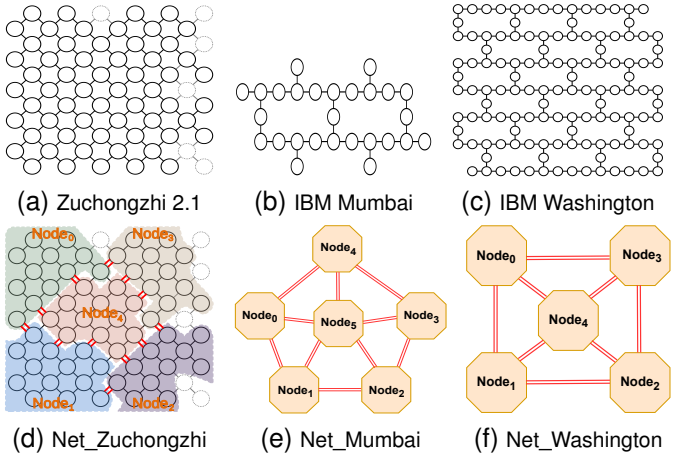


Fig. 5: The coupling graphs of three superconducting chips (a-c) and the DQC architectures (d-f) built using them.

a remote physical link, with each end of the connection typically linking one data qubit and two communication qubits (the device topology does not show communication qubits). (ii) ***Net_Mumbai*** (Fig. 5e): It is composed of six *IBMQ_Mumbai*. Red lines denote two remote physical connections between chips (communication qubits omitted). (iii) ***Net_Washington*** (Fig. 5f): It is composed of five *IBMQ_Washington*. Red lines represent three remote connections between chips, with communication qubits omitted from the topology.

Metrics: To evaluate our proposed algorithms, we use three metrics: (i) **estimated EPR pair usage** (lower is better): Approximates the number of EPR pairs consumed. We assume that executing a remote CNOT gate between adjacent chips consumes one EPR pair (using the *Cat-Comm* pattern), while executing a remote SWAP gate consume two EPR pairs (using the *TP-Comm* pattern). (ii) **estimated overall overhead** (lower is better): This metric comprehensively evaluates the execution time overhead caused by remote quantum communication and additional local SWAP gates in compiled quantum circuits. Initially, the overhead weight for local SWAP gates is set to 1. Moreover, based on the remote communication patterns for remote quantum gates, their structure, and the latency details of each atomic operation [14], we set the overhead weight of remote CNOT gates and remote SWAP gates (executed on adjacent quantum chips) to 10 and 20, respectively. (iii) **transpilation time** (lower is better): Execution time of the core part of qubit mapping algorithms or qubit routing algorithms.

Comparison baselines: For qubit mapping, we implement two state-of-the-art algorithms, named MHSA [23] and tket_dqc_map (qubit allocation method of tket_dqc [24]), as the comparison baselines, both accounting for chip capacity and inter-chip connectivity constraints. MHSA aims to allocate qubits to quantum chips while minimizing the estimated comprehensive overhead of remote quantum communication. tket_dqc_map is designed to distribute quantum circuits to DQC architectures while optimizing the EPR pair usage.

Since MHSA and tket_dqc_map do not address intra-chip qubit mapping, the following strategy is used to establish the initial mapping between logical and physical qubits within each chip. It aims to map frequently used logical qubits to highly connected physical qubits. Specifically, logical qubits are sorted in descending order based on the frequency of two-qubit gates acting on them, while physical qubits are sorted by the number of physical couplings they are connected to. The most frequently used logical qubit is then mapped to the most connected physical qubit, followed by a one-to-one mapping in order, ultimately obtaining the initial qubit placement.

For qubit routing, most existing algorithms [14], [24]–[26] for DQC focus solely on the routing of remote quantum gates. To our knowledge, there is currently no compilation algorithm that simultaneously takes into account the coordinated scheduling of both local and remote quantum gates. To evaluate our routing algorithm for co-scheduling of local and remote quantum gates, we choose the qubit routing method of SABRE [16] as the foundation and further train an improved version, refered as DSAR, as a comparative baseline.

To enable DSAR for DQC architectures and to minimize overall overhead as much as possible, we consider two key performance-related factors: (i) the time overhead ratio of executing a SWAP gate over a remote physical connection to that over a local connection, (ii) the lookahead ability, i.e., the maximum number of CNOT gates the extended set can hold. Accordingly, we set the value ranges for these two factors to $\{1, 10, 20, 30, 40, 50\}$ and $\{10, 20, 30, 40, 50\}$, respectively. We evaluate all 30 possible parameter combinations and, through comparison, select the optimal parameter combination that results in the least estimated overall overhead for each benchmark program. This optimized configuration is then applied to the routing method of SABRE to adapt and enhance it accordingly. In summary, DSAR is capable of coordinating the scheduling of the aforementioned two types of quantum gates while optimizing overall overhead.

To evaluate how DMapS performs in terms of remote communication overhead compared to existing DQC-oriented compilers, we choose tket_dqc [24] as a comparison baseline.

Algorithms setting: (i) MHSA: All parameters are consistent with the descriptions in [23]. (ii) tket_dqc: To address intra-chip qubit capacity constraints, tket_dqc_map employs a simulated annealing strategy for qubit placement. Meanwhile, during remote gate distribution, tket_dqc needs extra qubits

TABLE III: Simulation configuration summary

Questions	Algorithm Combinations	Benchmarks (Target Architectures)	Comparisons	Metrics	Results
Q1	$\mathbf{C}_1$: MHSA + DSAR $\mathbf{C}_2$: MHSA + DMapS-R $\mathbf{C}_3$: tket_dqc_map + DSAR $\mathbf{C}_4$: tket_dqc_map + DMapS-R $\mathbf{C}_5$: DMapS-M + DSAR $\mathbf{C}_6$: DMapS-M + DMapS-R	①: **Small-scale** (***Net_Zuchongzhi*** with intra-chip constrained connectivity) ②: **Medium-scale** (***Net_Mumbai*** with intra-chip constrained connectivity) ③: **Large-scale** (***Net_Washington*** with intra-chip constrained connectivity)	❶: $\mathbf{C}_1$ vs $\mathbf{C}_5$ ❷: $\mathbf{C}_2$ vs $\mathbf{C}_6$ ❸: $\mathbf{C}_3$ vs $\mathbf{C}_5$ ❹: $\mathbf{C}_4$ vs $\mathbf{C}_6$	1. EPR pair usage 2. Overall overhead 3. Transpilation time	Fig. 6
Q2			❶: $\mathbf{C}_1$ vs $\mathbf{C}_2$ ❷: $\mathbf{C}_3$ vs $\mathbf{C}_4$ ❸: $\mathbf{C}_5$ vs $\mathbf{C}_6$		Fig. 7
Q3			$\mathbf{C}_1$ v.s. $\mathbf{C}_2$ v.s. $\mathbf{C}_3$ v.s. $\mathbf{C}_4$ v.s. $\mathbf{C}_5$ v.s. $\mathbf{C}_6$		Tables IV-VI
Q4	$\mathbf{A}_1$: tket_dqc $\mathbf{A}_2$: DMapS	①, ②, and ③ are the same as above, ④: **Small-scale** (***Net_Zuchongzhi*** with intra-chip all-to-all connectivity) ⑤: **Medium-scale** (***Net_Mumbai*** with intra-chip all-to-all connectivity) ⑥: **Large-scale** (***Net_Washington*** with intra-chip all-to-all connectivity)	$\mathbf{A}_1$ vs $\mathbf{A}_2$	EPR pair usage	Fig. 8

on each chip for remote communication. Due to limited qubit resources, their number is set equal to the data qubits adjacent to communication qubits on related chips. (iii) DMapS-M: Iterative search rounds for the first stage are set to 20. For the second stage, W_1 is set to 0.5. (iv) DMapS-R: Set W_2, W_3, and W_r to 0.5, 0.7, and 2, respectively. For $Cost(\mathrm{SWAP})$, the weights of local and remote SWAP gates are set to 1 and 4, respectively. W_r and the weight of remote SWAP gates are set relatively low to avoid repetitive local searches.

Research questions: We conducted a series of experiments to answer the following research questions:

1) Q1. How does our qubit mapping algorithm (DMapS-M) compare with MHSA and tket_dqc_map?
2) Q2. How does our qubit routing algorithm (DMapS-R) compare with DSAR in co-scheduling local and remote quantum gates?
3) Q3. How does DMapS perform in co-scheduling local and remote quantum gates?
4) Q4. How does DMapS perform in terms of remote communication overhead compared to the existing DQC-oriented compiler (tket_dqc)?

Summary of simulation configuration: To present the simulation configuration in a more structured manner, Table III provides a concise summary of the compilation algorithm combinations, benchmark programs, comparative experiment design, and evaluation metrics for the four research questions.

B. Q1: Comparison of Mapping Algorithms

Experiment design: To answer Q1, we compare DMapS-M with MHSA and tket_dqc_map, each combined with the two qubit routing algorithms, DSAR and DMapS-R. For this purpose, four experimental settings are designed (see Table III), involving 108 runs over 27 benchmark programs. Due to nondeterminism in MHSA, tket_dqc_map, and DMapS-M, each experiment is repeated 10 times to compute the average EPR pair usage, overall overhead, and transpilation time. For MHSA, considering the increased runtime when dealing with large-scale qubits, we set a 24-hour transpilation time limit.

Result: Fig. 6 shows all comparative results, with 4 experiments marked "NA" as MHSA exceeded the 24-hour limit processing QFT_n320 and $Multiplier\_n400$. In Fig. 6(a)-6(i), all x-axes show the number of logical qubits. The y-axes indicate ratios of EPR pair usage (top, ΔEPR), overall overhead (middle, $\Delta Overall$), and transpilation time (bottom, Δt). Notably, exact values are displayed above the bars.

For instance, MHSA+DSAR and DMapS-M+DSAR incur EPR pair usage e_1, e_2 and overall overheads o_1, o_2, respectively. Then, $\Delta EPR = 1 - e_2/e_1$ and $\Delta Overall = 1 - o_2/o_1$. When $\Delta EPR > 0$, it indicates that DMapS-M has an improvement over the corresponding comparison algorithm, and the larger its value, the smaller the EPR pair usage introduced by DMapS-M. Similarly, $\Delta Overhead > 0$ conveys an analogous improvement in overall overhead.

The red dashed line represented by $y = 1$ indicates that DMapS-M has the same transpilation speed compared to other algorithms. Assuming the transpilation overheads of MHSA and DMapS-M are t_1 and t_2 respectively, the transpilation speed is calculated as t_1/t_2. The ratio $t_1/t_2 > 1$ indicates that DMapS-M has a smaller transpilation time.

Compared to MHSA, DMapS-M reduces EPR pair usage in 47/50 cases and overall overhead in 49/50 cases, while achieving faster transpilation in 49/50 results. On average (computed as the arithmetic mean across all experimental comparisons, the same method is used for subsequent averages), it lowers EPR pair usage by 41.35% (up to 98.25%), reduces overall overhead by 43.44% (up to 90.9%), and improves transpilation speed by 87.05x (up to 403.33x).

Compared to tket_dqc_map, DMapS-M achieves better EPR pair usage in 53/54 cases and lower overall overhead in all cases. On average, it reduces EPR pair usage by 63.35% (up to 99.06%) and overall overhead by 59.72% (up to 96.14%). However, its transpilation speed shows no clear advantage, especially for large-scale benchmarks (e.g., $Qugan\_n450$, $QAOA\_n400$). This is because the qubit grouping module in DMapS-M, which optimizes EPR pair overhead, and the intra-chip qubit mapping module, which minimizes local SWAP gate insertions, incur additional time overhead. In contrast, the simpler intra-chip qubit mapping strategy we added to tket_dqc_map leads to faster execution.

For specific benchmark programs (e.g., $Graphstate\_n500$, $Adder\_n433$, $Qugan\_n450$), DMapS-M achieves over 90% reduction in both EPR pair usage and overall overhead, primarily due to its enhanced performance in inter-chip qubit mapping. The qubit grouping module reduces the total number

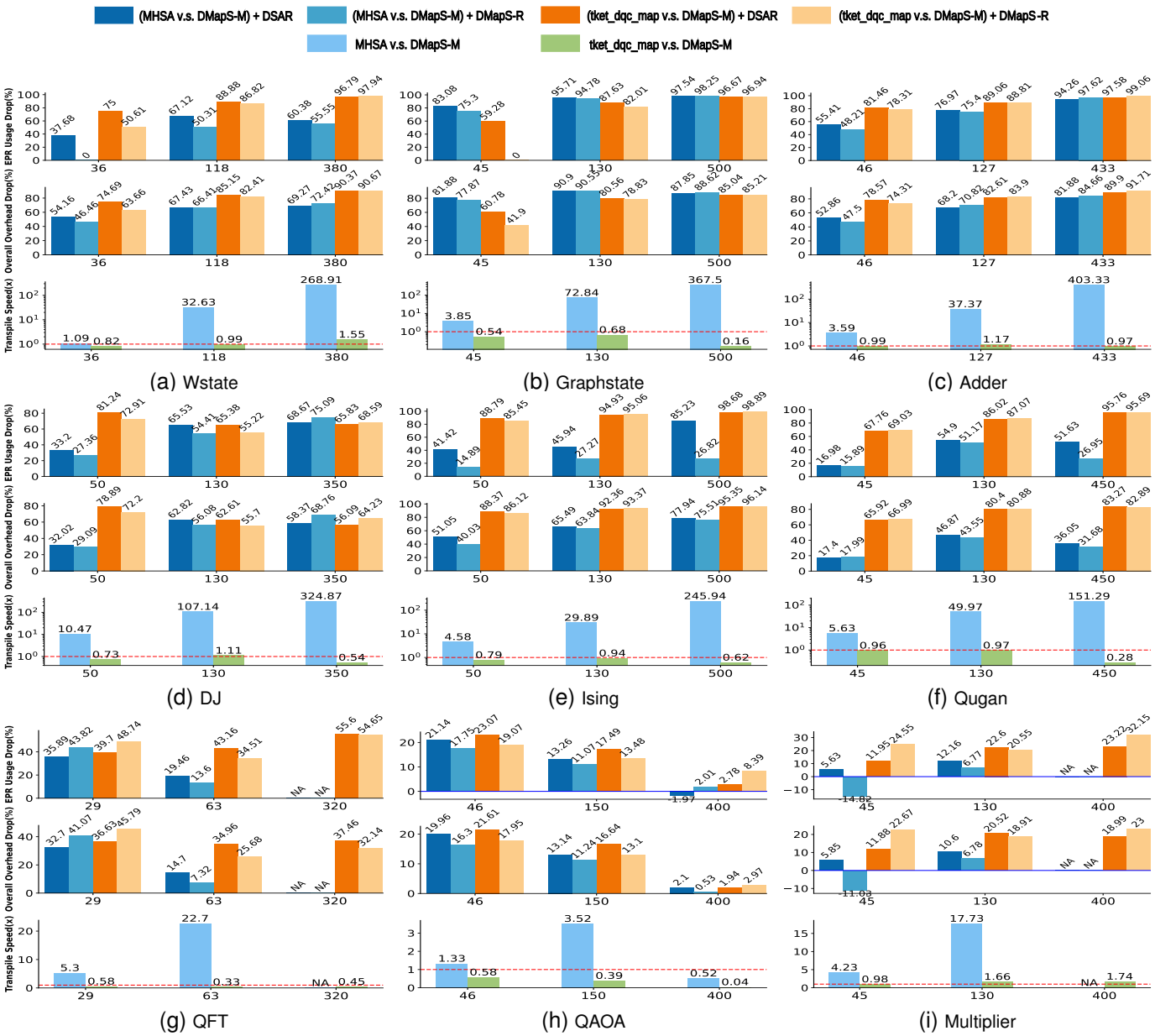


Fig. 6: The experimental comparison results of qubit mapping algorithms.

of remote quantum gates. Furthermore, the qubit group allocation module minimizes potential cross-chip remote quantum gates, thereby lowering communication overhead.

Summary: Compared to MHSA and tket_dqc_map, our qubit mapping algorithm DMapS-M significantly reduces both EPR pair usage and overall overhead. Moreover, DMapS-M also exhibits superior transpilation speed compared to MHSA. Furthermore, in terms of both optimization performance and transpilation speed, DMapS-M is capable of effectively handling large-scale qubit mapping problems.

C. Q2: Comparison of Routing Algorithms in Co-Scheduling Local and Remote Quantum Gates

Experiment design: To address Q2, we compare DMapS-R with DSAR, each integrated with three qubit mapping algorithms: MHSA, tket_dqc_map, and DMapS-M. Three types of experiments are conducted (see Table III), totaling 81 runs, each repeated 10 times to obtain average EPR pair usage, overall overhead, and transpilation time. For MHSA, a 24-hour time limit is imposed.

Result: Fig. 7 presents the comparative experimental results of qubit routing algorithms. Among them, two experiments did not yield results (marked with "NA"). Since MHSA failed to complete processing QFT_n320 and $Multiplier\_n400$ within the specified time, it was not possible to conduct comparisons between MHSA+DSAR and MHSA+DMapS-R.

In Fig. 7(a)-(i), the x-axes of the top, middle, and bottom charts indicate the number of qubits. Additionally, the y-axes of the top, middle, and bottom charts represent the reduction in EPR pair usage, overall overhead, and the ratio of transpilation time, respectively, calculated in the same manner as described in Section IV-B. The red dashed line at $y = 1$ denotes that DMapS-R and DSAR have equivalent transpilation speeds. Note that the transpilation time of qubit routing algorithms are the average of the time overheads when combined with three

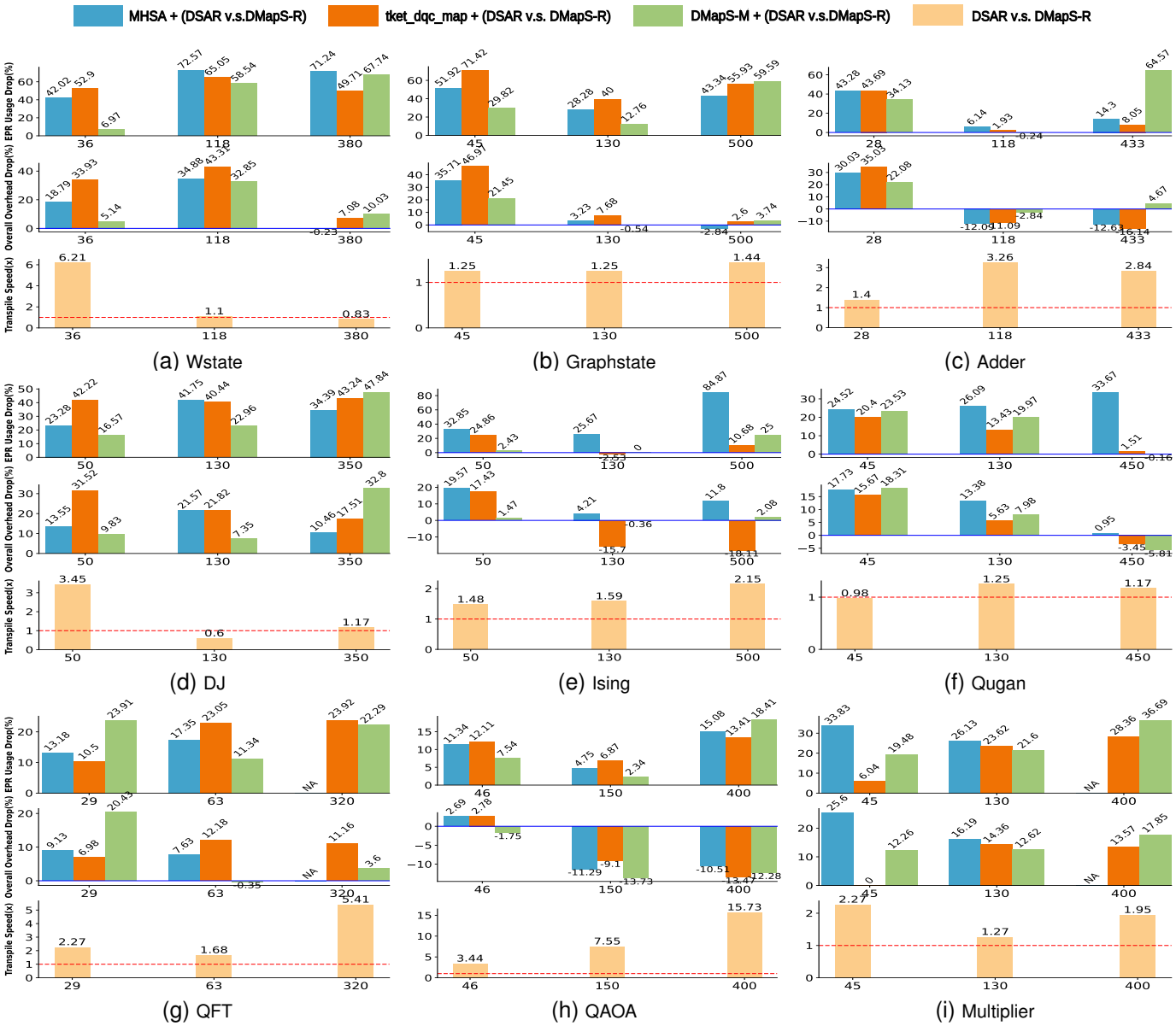


Fig. 7: The experimental comparison results between DSAR and DMapS-R.

distinct qubit mapping algorithms. In simple terms, a larger ΔEPR ($\Delta Overall$) means DMapS-R outperforms DSAR in optimizing EPR pair usage (overall overhead), and $\Delta t > 1$ indicates that DMapS-R executes faster than DSAR.

Out of all 79 valid experimental results, 75 results indicate that DMapS-R is superior to DSAR in terms of EPR pair usage. On average (the calculation method is the same as in Section IV-B), DMapS-R reduces EPR pair usage by 27.26% (with a maximum of 84.87%). Similarly, 57/79 of results show that DMapS-R delivers lower overall overhead compared to DSAR, with an average reduction of 8.85% (up to 46.97% reduction). Specifically, for quantum algorithms like QFT and $Multiplier$, which have very deep circuit depths, DMapS-R can still achieve considerable improvements over DSAR, even as the number of qubits gradually increases. An interesting phenomenon is that 17 experimental results show a positive improvement in EPR pair usage while exhibiting a negative improvement in overall overhead. This is primarily attributed to the local SWAP prioritization strategy employed in DMapS-R. Although this strategy achieves the goal of optimizing EPR pair usage, it also results in an excessive insertion of local SWAP gates. This drawback will be addressed in future improvements. In terms of transpilation speed, on average, DMapS-R is 2.78x faster than DSAR (up to 15.73x).

Summary: Compared to DSAR, when the qubit placements are already sufficiently good, DMapS-R can more effectively schedule remote and local quantum gates, often incurring less EPR pair usage and reduced overall overhead, and sometimes even faster execution speeds. Notably, DMapS-R performs well even when combined with different qubit mapping algorithms. Moreover, for larger-scale and more structurally complex quantum algorithms, DMapS-R also demonstrates outstanding performance. These two points indicate its excellent adaptability and scalability.

D. Q3: Overall Comparison in Co-Scheduling Local and Remote Quantum Gates

Experiment design: To answer Q3, we execute each of the

TABLE IV: Overall comparison (EPR pair usage)

Name	#Qubits	C_1	C_2	C_3	C_4	C_5	C_6
Wstate	36	6.9	**4** (✓)	17.2	8.1	4.3	**4** (✓)
	118	58.7	16.1	173.7	60.7	19.3	**8** (✓)
	380	31.3	9	387.4	194.8	12.4	**4** (✓)
Graphstate	45	33.7	16.2	14	**4** (✓)	5.7	**4** (✓)
	130	109.6	78.6	38	22.8	4.7	**4.1** (✓)
	500	403.5	228.6	297.3	131	9.9	**4** (✓)
Adder	46	84.1	47.7	202.3	113.9	37.5	**24.7** (✓)
	127	178.9	167.9	376.7	369.4	**41.2** (✓)	41.3
	433	472.6	405	1120.3	1030.1	27.1	**9.6** (✓)
DJ	50	26.2	20.1	93.3	53.9	17.5	**14.6** (✓)
	130	256.5	149.4	255.4	152.1	88.4	**68.1** (✓)
	350	517.8	339.7	474.7	269.4	162.2	**84.6** (✓)
Ising	50	14	9.4	73.2	55	8.2	**8** (✓)
	130	14.8	11	158	162	**8** (✓)	**8** (✓)
	500	54.2	8.2	608.4	543.4	8	**6** (✓)
Qugan	45	74.2	56	191.1	152.1	61.6	**47.1** (✓)
	130	173.2	128	558.8	483.7	78.1	**62.5** (✓)
	450	125.3	83.1	1431.6	1409.9	**60.6** (✓)	60.7
QFT	29	317	275.2	337	301.6	203.2	**154.6** (✓)
	63	654.4	540.8	927.2	713.4	527	**467.2** (✓)
	320	NA	NA	4556.4	3466.2	2022.8	**1571.8** (✓)
QAOA	46	722.8	640.8	741	651.2	570	**527** (✓)
	150	5429.4	5171.2	5707.6	5315.4	4709	**4598.4** (✓)
	400	23926	20316.2	25098.4	21730.4	24398.4	**19906.2** (✓)
Multiplier	45	872.8	**577.5** (✓)	935.4	878.9	823.6	663.1
	130	6985.2	5159.8	7927.3	6054.5	6135.4	**4810** (✓)
	400	NA	NA	39108.7	28015.3	30025.1	**19006.1** (✓)

* C_1: MHSA+DSAR, C_2: MHSA+DMapS-R, C_3: tket_dqc_map+DSAR, C_4: tket_dqc_map+DMapS-R, C_5: DMapS-M+DSAR, C_6: DMapS-M+DMapS-R
* ✓: the optimal algorithm combination, NA: blank experimental data caused by timeout

TABLE V: Overall comparison (Overall overhead)

Name	#Qubits	C_1	C_2	C_3	C_4	C_5	C_6
Wstate	36	13.57	11.02	24.58	16.24	6.22	**5.9** (✓)
	118	108.7	70.78	238.5	135.2	35.4	**23.77** (✓)
	380	272.77	273.4	870.36	808.73	83.81	**75.4** (✓)
Graphstate	45	39.37	25.31	18.18	9.64	7.13	**5.6** (✓)
	130	161.68	156.45	75.63	69.82	**14.7** (✓)	14.78
	500	841.42	865.37	683.42	665.59	102.23	**98.4** (✓)
Adder	46	99.62	69.7	219.23	142.43	46.96	**36.59** (✓)
	127	270.78	303.52	495.31	550.26	**86.1** (✓)	88.55
	433	1105.07	1244.73	1983.04	2303.21	200.19	**190.84** (✓)
DJ	50	32.01	27.67	103.11	70.6	21.76	**19.62** (✓)
	130	342.23	268.39	340.36	266.09	127.23	**117.87** (✓)
	350	1042.21	933.1	988.15	815.05	433.84	**291.5** (✓)
Ising	50	19.41	15.61	81.69	67.45	9.5	**9.36** (✓)
	130	47.5	45.5	214.56	248.25	**16.39** (✓)	16.45
	500	223.63	197.22	1061.77	1254.09	49.32	**48.29** (✓)
Qugan	45	86.45	71.12	209.53	176.68	71.4	**58.32** (✓)
	130	268.87	232.88	728.79	687.73	142.84	**131.44** (✓)
	450	711.48	704.68	2719.7	2813.55	**454.93** (✓)	481.38
QFT	29	351.65	319.52	373.43	347.35	236.63	**188.28** (✓)
	63	911.69	842.04	1195.78	1050.11	**777.62** (✓)	780.37
	320	NA	NA	7373.8	6550.6	4611.35	**4444.89** (✓)
QAOA	46	821.02	798.91	838.3	814.92	**657.09** (✓)	668.63
	150	7208.34	8022.76	7510.34	8194.52	**6260.53** (✓)	7120.3
	400	42505.21	46973.97	42437.48	48156.81	**41611.65** (✓)	46723.46
Multiplier	45	990.76	**737.1** (✓)	1058.65	1058.45	932.8	818.41
	130	8902.95	7460.83	10015.18	8576.26	7959.17	**6954.46** (✓)
	400	NA	NA	63201.16	54619.376	51193.94	**42054.46** (✓)

* C_1: MHSA+DSAR, C_2: MHSA+DMapS-R, C_3: tket_dqc_map+DSAR, C_4: tket_dqc_map+DMapS-R, C_5: DMapS-M+DSAR, C_6: DMapS-M+DMapS-R
* ✓: the optimal algorithm combination, NA: blank experimental data caused by timeout

six algorithm combinations (details are given in Table III) ten times for every benchmark and report the average EPR pair usage, overall overhead, and transpilation time. These metrics are used to evaluate which algorithm combination performs best in transpilation quality and speed.

Result: Table IV, Table V, and Table VI respectively present the average EPR pair usage, overall overhead, and transpilation time results for different algorithm combinations. The "NA" symbol denotes results were not obtained due to MHSA failing to terminate within the execution time limit. In Table IV and Table V, the best algorithm combination is marked with ✓.

In Table IV, regarding EPR pair usage, 24/27 of benchmark programs indicate that DMapS-M+DMapS-R performs optimally. For the remaining three, using our qubit mapping algorithm alone can also achieve optimal results. Similarly, Table V indicates that DMapS-M+DMapS-R yields optimal overall overhead in 18 out of 27 cases, while for the other nine benchmarks, either our qubit mapping or qubit routing

TABLE VI: Overall comparison (Transpilation time (s))

Name	#Qubits	C_1	C_2	C_3	C_4	C_5	C_6
Wstate	36	3.69	3.67	3.72	**2.81**	3.35	3.35
	118	155.09	154.7	5.85	5.9	**5.12**	5.24
	380	3234.91	3235.57	23.16	23.8	**13.75**	14.38
Graphstate	45	13.76	13.7	2.05	**2.04**	3.56	3.57
	130	311.95	309.25	6.45	6.9	**4.89**	5.24
	500	17119.03	16806.86	442.62	355.77	**123.01**	191.19
Adder	46	16.53	16.25	5.03	4.85	4.65	**4.61**
	127	314.95	307.19	41.63	15.66	10.37	**8.96**
	433	25349.66	24431.25	2885.36	592.82	75.18	**70.64**
DJ	50	41.53	40.98	**2.94**	2.97	3.95	3.95
	130	507.4	507.78	5.65	5.79	**4.93**	5
	350	7912.48	7912.13	17.06	**16.66**	27.25	26.45
Ising	50	16.57	16.58	3.49	**3.14**	3.61	3.62
	130	162.16	163.83	19.95	9.41	**6.6**	6.89
	500	8241.72	8373.91	1296.19	259.67	**97.89**	131.99
Qugan	45	29.76	29.75	5.85	5.69	**5.48**	5.68
	130	659.83	659.49	16.54	15.27	**14.32**	14.51
	450	39019.74	39023.64	105.37	**91.59**	270.65	269.9
QFT	29	56.38	55.53	8.1	**6.88**	11.81	10.88
	63	1158.46	1155.22	24.75	**21.74**	57.74	54.91
	320	NA	NA	563.65	**314.83**	851.98	660.02
QAOA	46	31.16	23.02	21.51	**12.28**	24.6	17.93
	150	1563.13	939.62	853.58	**189.26**	796.69	324.25
	400	53063.99	23311.08	35431.7	**3479.36**	53211.5	42637.96
Multiplier	45	65.25	62.07	20.01	16.86	18.51	**16.16**
	130	3409.96	3403.18	366.96	348.59	218.42	**218.26**
	400	NA	NA	26590.97	25566.17	14748.31	**14680.4**

* C_1: MHSA+DSAR, C_2: MHSA+DMapS-R, C_3: tket_dqc_map+DSAR, C_4: tket_dqc_map+DMapS-R, C_5: DMapS-M+DSAR, C_6: DMapS-M+DMapS-R
* NA: blank experimental data caused by timeout

algorithm alone achieves optimal outcomes. Overall, given the already strong performance of DMapS-M, combining it with DMapS-R reveals significant additional optimization potential.

As mentioned above, regarding EPR pair usage and overall overhead, the optimal algorithm combinations appeared with the following frequencies: DMapS-M+DMapS-R (42/54), DMapS-M+DSAR (11/54), MHSA+DMapS-R (3/54), and tket_dqc_map+DMapS-R (1/54). Among these optimal combinations, DMapS-M+DMapS-R achieves the fastest transpilation in 6 out of 27 cases and ranks second in 17 cases (see Table VI). Notably, although tket_dqc_map+DSAR and tket_dqc_map+DMapS-R have the least transpilation time in some cases (mainly because tket_dqc_map itself requires less execution time, as explained in Section IV-B), their performance in other key metrics is less effective.

Summary: When DMapS-M and DMapS-R are combined for use in DQC architectures, they often outperform other algorithm combinations in terms of EPR pair usage and overall overhead, while also achieving excellent transpilation speed. Additionally, the improvements in EPR pair usage and overall overhead achieved by this combination, as well as those obtained by using DMapS-M or DMapS-R individually, surpass the gains obtained by using the baseline algorithms.

E. Q4: Comparison of tket_dqc and DMapS in Terms of Remote Quantum Communication Overhead

Experiment design: To address Q4, we configure tket_dqc and DMapS with their respective qubit mapping and routing algorithms, and execute each benchmark 10 times on the corresponding DQC architectures (where the intra-chip topology includes both constrained and all-to-all connectivity types). Average EPR pair usage serves as the evaluation metric. Further details are provided in Table III.

Result: Fig. 8 shows the average EPR pair usage of tket_dqc and DMapS when executing various benchmark programs on DQC architectures with different intra-chip connectivity settings. In Fig. 8(a)-(b), the x-axis represents the number of logical qubits, while the y-axis indicates EPR pair usage

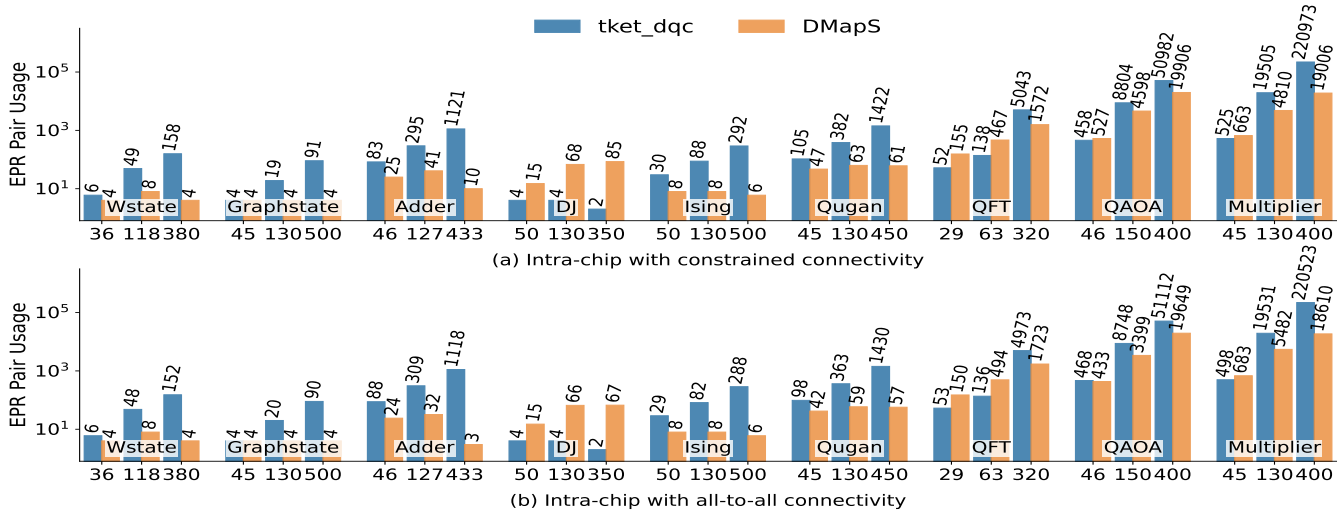


Fig. 8: Experimental comparison of EPR pair usage by tket_dqc and DMapS under two intra-chip connectivity settings in DQC architectures: (a) constrained connectivity and (b) all-to-all connectivity.

quantities. Smaller EPR pair consumption values at polyline vertices indicate superior algorithm performance, and ΔEPR quantifies the relative reduction in EPR pair usage. Given that the EPR pair usage introduced by tket_dqc and DMapS is denoted by e_1 and e_2 respectively, then $\Delta EPR = 1 - e_2/e_1$.

Among the 54 experimental cases conducted based on 27 benchmark programs, DMapS outperformed tket_dqc in 39/54 cases. In these superior cases, DMapS achieved an average 75.16% reduction in EPR pair usage (up to 99.73%) compared to tket_dqc. For certain quantum circuits (e.g., DJ), the frequent sharing of a qubit among multiple two-qubit gates allows tket_dqc to fully exploit its shared EPR pair mechanism, thereby enabling efficient execution of multiple remote quantum gates. Experimental results show that DMapS increases more EPR pair usage in these benchmark programs.

Summary: tket_dqc introduces additional physical qubits to enable EPR pair reuse. In contrast, DMapS incurs higher remote communication overhead due to its strict adherence to inter-chip connectivity constraints and the limited qubit capacity within each chip. Nevertheless, experimental results demonstrate that DMapS still shows significant advantages in EPR pair usage for certain quantum programs.

V. Related Work

For qubit mapping and routing on single superconducting chips, prior works improve circuit execution by reducing circuit size [15], [16], [18], [19] and enhancing fidelity [17], etc. In DQC, research increasingly targets reducing remote communication overhead, e.g., by partitioning circuits to minimize remote gates [20], [22], though often under idealized architectures with full inter-chip connectivity. More recent methods [23], [24] perform qubit-to-chip mapping under chip capacity and inter-chip connectivity constraints. Our approach further maps logical to physical qubits while jointly optimizing EPR pair usage and local SWAP insertion.

Prior works [14], [24]–[26] optimize remote gate scheduling by reducing EPR pair usage and time overhead. However, intra-chip topology constraints in near-term devices require joint scheduling of local and remote gates. To address this, we propose a new qubit routing algorithm that not only achieves coordinated scheduling of local and remote quantum gates but also realizes some performance advantages.

VI. Conclsion and Future Work

In this paper, we present the first end-to-end qubit mapping and routing algorithms for near-term DQC architectures, optimizing EPR pair overhead and local SWAP insertions to improve execution efficiency. The mapping algorithm decomposes large-scale tasks, incrementally reduces overhead, and accelerates processing via parallelized intra-chip qubit mapping. The routing algorithm reduces EPR pair usage by prioritizing local SWAP gates and enhances transpilation speed through parallelized intra-chip qubit routing.

For future work, we plan to design more efficient distributed compilation algorithms tailored to specific applications (such as quantum machine learning) and to distributed quantum computing architectures based on alternative technological platforms (e.g., neutral atoms, ion traps). Furthermore, we intend to develop more robust quantum compilation algorithms using constraint solving techniques.

References

[1] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge University Press, 2010.

[2] S. Lloyd, "Universal quantum simulators," *Science*, vol. 273, pp. 1073–1078, 1996.

[3] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci. (FOCS)*, 1994, pp. 124–134.

[4] G. Q. AI *et al.*, "Quantum error correction below the surface code threshold," *Nature*, vol. 638, no. 8052, p. 920, 2024.

[5] N. P. De Leon, K. M. Itoh, D. Kim, K. K. Mehta, T. E. Northup, H. Paik, B. Palmer, N. Samarth, S. Sangtawesin, and D. W. Steuerman, "Materials challenges and opportunities for quantum computing hardware," *Science*, vol. 372, no. 6539, p. eabb2823, 2021.

[6] P. Zhao, K. Linghu, Z. Li, P. Xu, R. Wang, G. Xue, Y. Jin, and H. Yu, "Quantum crosstalk analysis for simultaneous gate operations on superconducting qubits," *PRX quantum*, vol. 3, no. 2, p. 020301, 2022.

[7] Y. Zhong, H.-S. Chang, A. Bienfait, É. Dumur, M.-H. Chou, C. R. Conner, J. Grebel, R. G. Povey, H. Yan, D. I. Schuster *et al.*, "Deterministic multi-qubit entanglement in a quantum network," *Nature*, vol. 590, no. 7847, pp. 571–575, 2021.

[8] J. Niu, L. Zhang, Y. Liu, J. Qiu, W. Huang, J. Huang, H. Jia, J. Liu, Z. Tao, W. Wei *et al.*, "Low-loss interconnects for modular superconducting quantum processors," *Nat. Electron.*, vol. 6, no. 3, pp. 235–241, 2023.

[9] D. Main, P. Drmota, D. Nadlinger, E. Ainley, A. Agrawal, B. Nichol, R. Srinivas, G. Araneda, and D. Lucas, "Distributed quantum computing across an optical network link," *Nature*, pp. 1–6, 2025.

[10] K. M. Svore, A. V. Aho, A. W. Cross, I. Chuang, and I. L. Markov, "A layered software architecture for quantum computing design tools," *Computer*, no. 1, pp. 74–83, 2006.

[11] A. G. Fowler, S. J. Devitt, and L. C. L. Hollenberg, "Implementation of shor's algorithm on a linear nearest neighbour qubit array," *Quantum Inf. Comput.*, vol. 4, no. 4, pp. 237–251, 2004.

[12] A. Shafaei, M. Saeedi, and M. Pedram, "Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures," in *Proc. 50th Annu. Des. Autom. Conf. (DAC)*, 2013, pp. 1–6.

[13] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?" *Phys. Rev.*, vol. 47, no. 10, p. 777, 1935.

[14] 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. 55th IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2022, pp. 1027–1041.

[15] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 7, pp. 1226–1236, 2018.

[16] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for NISQ-era quantum devices," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2019, pp. 1001–1014.

[17] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2019, pp. 1015–1029.

[18] L. Lao and D. E. Browne, "2qan: A quantum compiler for 2-local qubit hamiltonian simulation algorithms," in *Proc. 49th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2022, pp. 351–365.

[19] C. Luo, S. Cao, S. Guo, and C. Hu, "Towards effective local search for qubit mapping," *IEEE Trans. Comput.*, 2025.

[20] P. Andrés-Martínez and C. Heunen, "Automated distribution of quantum circuits via hypergraph partitioning," *Phys. Rev. A*, vol. 100, p. 032308, 2019.

[21] J.-Y. Wu, K. Matsui, T. Forrer, A. Soeda, P. Andrés-Martínez, D. Mills, L. Henaut, and M. Murao, "Entanglement-efficient bipartite-distributed quantum computing," *Quantum*, vol. 7, p. 1196, 2023.

[22] P. Escofet, A. Ovide, M. Bandic, L. Prielinger, H. Van Someren, S. Feld, E. Alarcon, S. Abadal, and C. Almudever, "Revisiting the mapping of quantum circuits: Entering the multi-core era," *ACM Trans. Quantum Comput.*, vol. 6, no. 1, pp. 1–26, 2025.

[23] Y. Mao, Y. Liu, and Y. Yang, "Qubit allocation for distributed quantum computing," in *Proc. 42nd IEEE Conf. Comput. Commun. (INFOCOM)*, 2023, pp. 1–10.

[24] P. Andres-Martinez, T. Forrer, D. Mills, J.-Y. Wu, L. Henaut, K. Yamamoto, M. Murao, and R. Duncan, "Distributing circuits over heterogeneous, modular quantum computing network architectures," *Quantum Sci. Technol.*, vol. 9, no. 4, p. 045021, 2024.

[25] D. Ferrari, A. S. Cacciapuoti, M. Amoretti, and M. Caleffi, "Compiler design for distributed quantum computing," *IEEE Trans. Quantum Eng.*, vol. 2, pp. 1–20, 2021.

[26] D. Ferrari, S. Carretta, and M. Amoretti, "A modular quantum compilation framework for distributed quantum computing," *IEEE Trans. Quantum Eng.*, 2023.

[27] X. Deng, W. Zheng, X. Liao, H. Zhou, Y. Ge, J. Zhao, D. Lan, X. Tan, Y. Zhang, S. Li *et al.*, "Long-range zz interaction via resonator-induced phase in superconducting qubits," *Phys. Rev. Lett.*, vol. 134, no. 2, p. 020801, 2025.

[28] H. Yan, Y. Zhong, H.-S. Chang, A. Bienfait, M.-H. Chou, C. R. Conner, É. Dumur, J. Grebel, R. G. Povey, and A. N. Cleland, "Entanglement purification and protection in a superconducting quantum network," *Phys. Rev. Lett.*, vol. 128, no. 8, p. 080504, 2022.

[29] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels," *Phys. Rev. Lett.*, vol. 70, p. 1895, 1993.

[30] R. W. Floyd, "Algorithm 97: shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, 1962.

[31] L. Gottesbüren, M. Hamann, S. Schlag, and D. Wagner, "Advanced flow-based multilevel hypergraph partitioning," in *Proc. 18th Int. Symp. Exp. Algorithms (SEA)*, 2020.

[32] V. Chaudhary and J. K. Aggarwal, "A generalized scheme for mapping parallel algorithms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 3, pp. 328–346, 1993.

[33] F. Glover, "Tabu search—part I," *OORSA J. Comput.*, vol. 1, no. 3, pp. 190–206, 1989.

[34] S. Schlag, "High-quality hypergraph partitioning," Ph.D. dissertation, Karlsruhe Institute of Technology, Germany, 2020.

[35] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "QASMBench: A low-level quantum benchmark suite for NISQ evaluation and simulation," *ACM Trans. Quantum Comput.*, vol. 4, no. 2, pp. 1–26, 2023.

[36] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking software and design automation tools for quantum computing," *Quantum*, vol. 7, p. 1062, 2023.

[37] Q. Zhu, S. Cao, F. Chen, M.-C. Chen, X. Chen, T.-H. Chung, H. Deng, Y. Du, D. Fan, M. Gong *et al.*, "Quantum computational advantage via 60-qubit 24-cycle random circuit sampling," *Sci. Bull.*, vol. 67, no. 3, pp. 240–245, 2022.

[38] J. Gambetta, "IBM's roadmap for scaling quantum technology," https://www.ibm.com/quantum/blog/ibm-quantum-roadmap?mhsrc=ibmsearch_a&mhq=condor, 2024.

[39] J. Chow, O. Dial, and J. Gambetta, "IBM quantum breaks the 100-qubit processor barrier," https://www.ibm.com/quantum/blog/127-qubit-quantum-processor-eagle?social_post=5922821977&linkId=140350920, 2021.

Tingyu Luo received his B.E. degree in computer science and technology from Anhui Agricultural University, China. He is currently pursuing the Ph.D. degree with the Software Engineering Institute, East China Normal University, Shanghai, China.

His current research interests include distributed quantum computing, quantum compilation, and quantum architecture.

Yuzhen Zheng received the B.S. degree in computer science and technology from Sichuan University, China, in 2019, and the M.S. degree in computer science and technology from the National University of Defense Technology, China, in 2022.

His current research interests include quantum computing, quantum compilation, and quantum programming theory.

Yuxin Deng received his B.Eng. degree in thermal energy engineering and M.Sc. degree in computer science from Shanghai Jiao Tong University, Shanghai, in 1999 and 2002, respectively, and Ph.D. degree in computer science from Ecole des Mines de Paris, Paris, in 2005.

He is currently a professor at Shanghai University of Finance and Economics, Shanghai. His research interests include concurrency theory, especially about process calculi, formal semantics of programming languages, as well as quantum computing. He authored the book titled Semantics of Probabilistic Processes: An Operational Approach (Springer, 2015).

Xiang Fu received his B.E., M.E. and Ph.D. degree from Tsinghua University, National University of Defense Technology and Delft University of Technology in 2011, 2014, and 2018, respectively.

He is currently an Associate Professor with College of Computer Science and Technology, National University of Defense Technology. His current research interests include quantum programming and compilation, quantum control software, and quantum control (micro)architecture.